68' MICRO JOURNAL

Australia A $ 4.75 Singapore S $ 9.45 Malaysia M $ 9.45 New Zealand NZ $ 6.50 Hong Kong H $23.50 Sweden 30/-SEK

$2.95 USA

Motorola VME-MACINTOSH-S 50 & Other 68XXX Systems
6809 68008 68000 68010 68020 68030

OS-9 The Magazine for Motorola CPU Devices FLEX SK*DOS
A User Contributor Journal

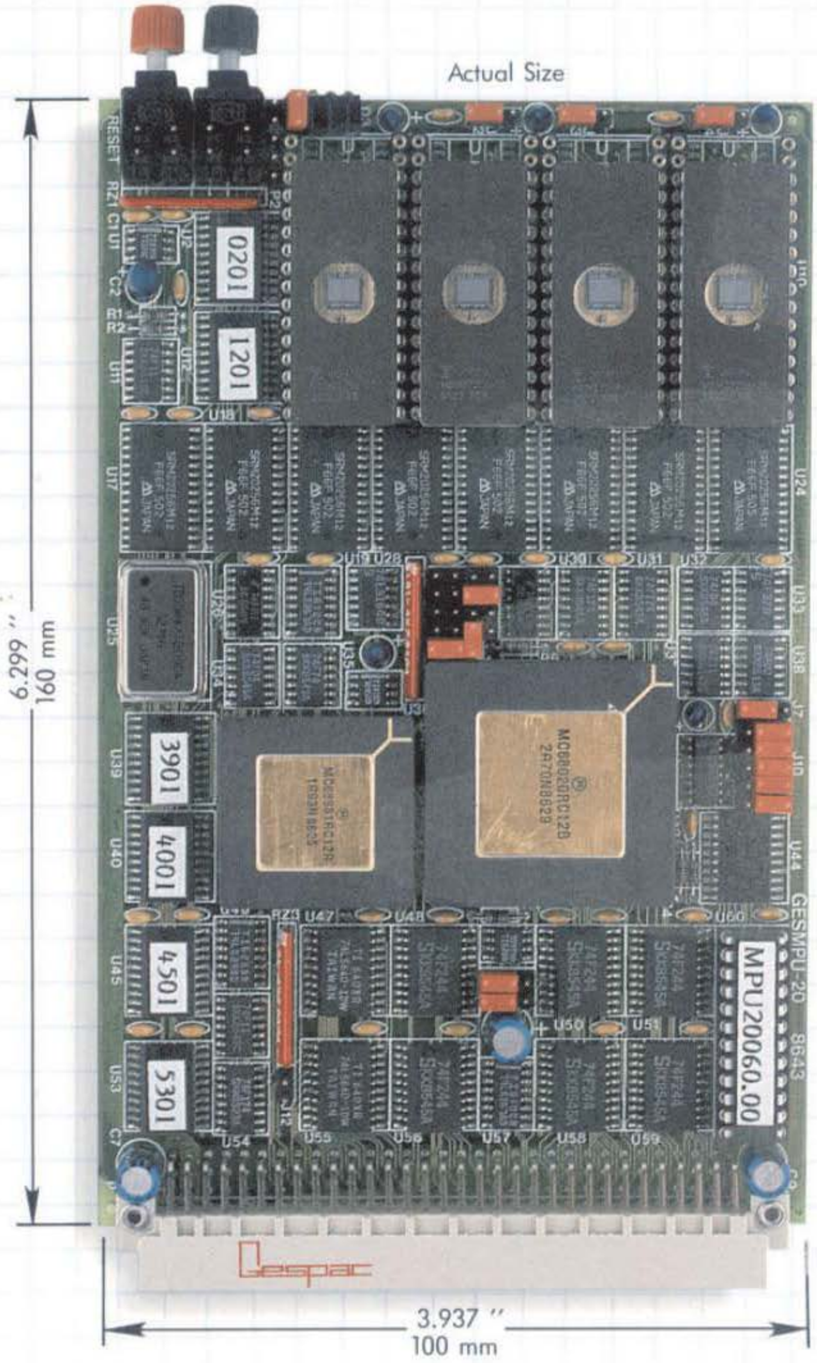
This Issue:

VOLUME X ISSUE 1 ● Devoted to the 68XXX User ● January 1988

The Grandfather of "DeskTop Publishing™"

PHOTO CREDIT: NASA

GESPAC Gives You More Power Per Square Inch.



Here is the size, performance, and cost breakthrough you have been waiting for: The 68020 based GESMPU-20 from GESPAC.

You can now unleash an unprecedented amount of power into your applications. On just 25 square inches, we have squeezed a 12.5 MHz (16 MHz optional) 68020 32-bit microprocessor, a 68881 floating point coprocessor, 4 sockets for up to 512 Kilobytes of EPROM, and up to 512 Kilobytes of zero-wait-states CMOS RAM.

This board is totally expandable through its G-64 bus interface. And GESPAC has the largest variety of inexpensive memory, interfaces, controllers, and transducer cards anywhere. Plus real-time disk operating systems, high level languages, and other software tools. GESPAC has the total solution to your system integration needs.

Best of all, because our boards are small, they cost less. The new GESMPU-20 is priced below $1000 in one hundred piece quantity orders.

So why wait? Contact us today for information on the GESMPU-20 or any of the 150 G-64 bus system components from GESPAC—the leader in single Eurocard microcomputer products worldwide.

Call (602) 962-5559.

Gespac

IN USA - CANADA
50A West Hoover Ave.
Mesa, Arizona 85202
Tel. (602) 962-5559
Telex 386575

INTERNATIONAL
3, chemin des Aulx
CH-1228 Geneva
Tel. (022) 713400
Telex 429989

GMX MICRO-20 PRICE LIST

MICRO 20 (12.5 MHz) W/1 SAB	$2565.00
MICRO 20 (16.67 MHz) W/1 SAB	$2895.00
MICRO 20 (20 MHz) W/1 SAB	$3295.00

OPTIONAL PARTS AND ACCESSORIES

68881 12.5MHz Floating Point Coprocessor	$ 195.00
68881 16.67MHz Floating Point Coprocessor	$ 295.00
68881 20MHz Floating Point Coprocessor	$ 495.00
MOTOROLA 68020 USERS MANUAL	$ 18.00
MOTOROLA 68881 USERS MANUAL	$ 18.00

SBC ACCESSORY PACKAGE (M20-AP) $1690.00

The package includes a PC-style cabinet with a custom backpanel, a 25 Megabyte (unformatted) hard disk and controller, a floppy disk drive, a 150 watt power supply, cooling fan, panel mounted reset and abort switches, and all necessary internal cabling. (For use with SAB—9D serial connectors only.)

2nd 5"80 FLOPPY & CABLES FOR M20-AP, ADD	$ 250.00
SECOND 25MB HARD DISK & CABLES, ADD	$ 780.00
TO SUBSTITUTE 50MB HD FOR 25MB HD, ADD	$ 290.00
TO SUBSTITUTE 80MB HD FOR 25MB HD, ADD	$1500.00
TO SUBSTITUTE 155MB FOR 25MB HD, ADD	$2100.00
60MB TEAC STREAMER WITH ONE TAPE	$ 870.00
PKG. OF 5 TEAC TAPES	$ 112.50
CUSTOM BACK PANEL PLATE (BPP-PC)	$ 44.00

I/O EXPANSION BOARDS

16 PORT SERIAL BOARD ONLY (SBC-16S) .... $ 335.00

The SBC-16S extends the I/O capabilities of the GMX Micro-20 68020 Single-board Computer by adding sixteen asynchronous serial I/O ports. By using two SBC-16S boards, a total of thirty-six serial ports are possible.

RS232 ADAPTER (SAB-25, SAB-9D or SAB-8M) $165.00

The board provides level-shifting between TTL level and standard RS-232 signal levels for up to 4 serial I/O ports.

60 LINE PARALLEL I/O BOARD (SBC-60P) $398.00

The GMX SBC-60P uses three 68230 Parallel Interface/Timers (PI/Ts) to provide up to forty-eight parallel I/O lines. The I/O lines are buffered in six groups of eight lines each, with separate buffer direction control for each group. Buffer direction can be fixed by hardware jumpers, or can be software programmable for bidirectional applications.

PROTOTYPING BOARD (SBC-WW) $75.00

The SBC-WW provides a means of developing and testing custom I/O interface designs for the GMX Micro-20 68020 Single-board Computer. The board provides areas for both DIP (Dual Inline Package) and PGA (Pin Grid Array) devices, and a pre-wired memory area for up to 512K bytes of dynamic RAM.

I/O BUS ADAPTER (SBC-BA) $195.00

The SBC-BA provides an interface between the GMX Micro-20 68020 Single-board Computer and the Motorola Input/Output Channel (I/O bus). With the I/O bus, up to sixteen off-the-shelf or custom peripheral devices (I/O modules) can be connected to the GMX Micro-20.

ARCNET LAN board w/o Software (SBC-AN) $475.00

The SBC-AN provides an interface between the GMX Micro-20 68020 Single-board Computer and the ARCNET modified token-passing Local Area Network (LAN) originally developed by Datapoint Corp. The ARCNET is a baseband network with a data transmission rate of 2.5 Megabits/second. The standard transmission media is a single 93 ohm RG-62/U coaxial cable. Fiber optic versions are available as an option.

OS9 LAN Software Drivers for SBC-AN $120.00

GMX MICRO-20 SOFTWARE

020 BUG UPDATE — PROMS & MANUAL $150.00

THESE 68020 OPERATING SYSTEMS ARE PRICED WHEN PURCHASED WITH THE MICRO-20. PLEASE ADD $150.00 IF PURCHASED LATER FOR THE UPDATED PROMS AND MANUALS. ALL SHIPPED STANDARD ON 5¼" DISKS. 3½" OPTIONAL IF SPECIFIED.

OS9/68020 PROFESSIONAL PAK $850.00

Includes O.S., "C", uMACS EDITOR, ASSEMBLER, DEBUGGER, development utilities, 68881 support.

OS9/68020 PERSONAL PAK .... $ 400.00

Personal OS-9 systems require a GMX Micro-20 development system running Professional OS-9/68020 for initial configuration.

Other Software for OS-9/68020

BASIC (Included in PERSONAL PAK)	$ 200.00
C COMPILER (included in PROFESSIONAL PAK)	$ 750.00
PASCAL COMPILER	$ 500.00

UNIFLEX

UniFLEX	$ 450.00
UniFLEX WITH REAL-TIME ENHANCEMENTS	$ 800.00

Other Software for UniFLEX

UniFLEX BASIC W/PRECOMPILER	$ 300.00
UniFLEX C COMPILER	$ 350.00
UniFLEX COBOL COMPILER	$ 750.00
UniFLEX SCREEN EDITOR	$ 150.00
UniFLEX TEXT PROCESSOR	$ 200.00
UniFLEX SORT/MERGE PACKAGE	$ 200.00
UniFLEX VSAM MODULE	$ 100.00
UniFLEX UTILITIES PACKAGE I	$ 200.00
UniFLEX PARTIAL SOURCE LICENSE	$1000.00

GMX EXCLUSIVE VERSIONS, CUSTOMIZED FOR THE MICRO-20, OF THE BELOW LANGUAGES AND SOFTWARE ARE ALSO AVAILABLE FROM GMX.

ABSOFT FORTRAN (UniFLEX)	$1500.00
SCULPTOR (specify UniFLEX or OS9)	$ 995.00
FORTH (OS9)	$ 595.00
DYNACALC (specify UniFLEX or OS9)	$ 300.00

GMX DOES NOT GUARANTEE PERFORMANCE OF ANY GMX SYSTEMS, BOARDS OR SOFTWARE WHEN USED WITH OTHER MANUFACTURERS PRODUCT.

ALL PRICES ARE F.O.B. CHICAGO IN U.S. FUNDS

TO ORDER BY MAIL: SEND CHECK OR MONEY ORDER OR USE YOUR VISA OR MASTER CHARGE. Please allow 3 weeks for personal checks to clear. U.S. orders add $5 handling if under $200.00. Foreign orders add $10 handling if order is under $200.00. Foreign orders over $200.00 will be shipped via Emery Air Freight COLLECT, and we will charge no handling. All orders must be prepaid in U.S. funds. Please note that foreign checks have been taking about 8 weeks for collection so we would advise wiring money, or checks drawn on a bank account in the U.S. Our bank is the Continental Illinois National Bank of Chicago, 231 S. LaSalle Street, Chicago, IL 60693, account number 73-32033.

CONTACT GMX FOR MORE INFORMATION ON THE ABOVE PRODUCTS

GMX STILL SELLS GIMIX S50 BUS SYSTEMS, BOARDS & PARTS. CONTACT GMX FOR COMPLETE PRICE LIST.

GMX 1337 W. 37th Place, Chicago, IL 60609 (312) 927-5510 — TWX 910-221-4055 — FAX (312) 927-7352

A Member of the CPI Family

68 Micro Journal

10 Years of Dedication to Motorola CPU Users

6800 6809 68000 68010 68020

The Originator of "DeskTop Publishing™"

Publisher
Don Williams Sr.

Executive Editor
Larry Williams

Production Manager
Tom Williams

Office Manager
Joyce Williams

Subscriptions
Stacy Power

Contributing & Associate Editors

Ron Anderson
Ron Voigts
Doug Lurie
Ed Law

Dr. E.M. "Bud" Pass
Art Weller
Dr. Theo Elbert
& Hundreds More of Us

Contents

68 MICRO JOURNAL

"Contribute Nothing - Expect Nothing" DMW 1986

COMPUTER PUBLISHING, INC.

"Over a Decade of Service"

"World Wide"

68 MICRO JOURNAL
Computer Publishing Center
5900 Cassandra Smith Road
PO Box 849
Hixson, TN 37343

Phone (615) 842-4600 Telex 510 600-6630

68 Micro Journal is the *original* "DeskTop Publishing" product and has continously published since 1978 using only micro-computers and special "DeskTop" software. Using first a kit built 6800 micro-computer, a modified "ball" typewriter, and "home grown" DeskTop Publishing software. None was commercially available at that time. For over 10 years we have been doing "DeskTop Publishing"! *We originated what has become traditional "DeskTop Publishing"!* Today 68 Micro Journal is acknowledged as the "Grandfather" of "DeskTop Publishing" technology.

68 Micro Journal is published 12 times a year by Computer Publishing Inc. Second Class Postage paid ISSN 0194-5025 at Hixson, TN. and additional entries. Postmaster: send form 3597 to 68 Micro Journal, POB 849, Hixson, TN 37343.

Subscription Rates

1 Year $24.50 USA, Canada & Mexico $34.00 a year. Others add $12.00 a year surface, $48.00 a year Airmail, USA funds. 2 years $42.50, 3 years $64.50 plus additional postage for each additional year.

Items or Articles for Publication

Articles submitted for publication must include authors name, address, telephone number, date and a statement that the material is original and the property of the author. Articles submitted should be on diskette, OS-9, SK*DOS, FLEX, Macintosh or MS-DOS. All printed items should be dark type and satisfactory for photo-reproduction. No blue ink! No hand written articles - please! Diagrams o.k.

Please - do not format with spaces any text indents, charts, etc. (source listing o.k.). We will edit in all formatting. Text should fall flush left and use a carriage return only to indicate a paragraph end. Please write for free authors guide.

Letters & Advertising Copy

Letters to the Editor should be the original copy, signed! Letters of grip as well as praise are acceptable. *We reserve the right to reject any letter or advertising material, for any reason we deem advisable.* Advertising Rates: Commercial please contact 68 Micro Journal Advertising Department. Classified advertising must be non-commercial. Minimum of $15.50 for first 15 words. Add $.60 per word thereafter. No classifieds accepted by telephone.

MUSTANG -020 Super SBC ™

DATA-COMP Proudly Presents the First Under $4300 "SUPER MICRO" See other advertising (backcover) for economy system (68008) - under $2400 complete!

CPI ™
Mustang Hi-Speed Systems
Only from Data-Comp Div.
68008-68030

The MUSTANG-020 68020 SBC provides a powerful, compact, 32 bit computer system featuring the "state of the art" Motorola 68020 "super" micro-processor. It comes standard with 2 megabyte of high-speed SIP dynamic RAM, serial and parallel ports, floppy disk controller, a SASI hard disk interface for intelligent hard disk controllers and a battery backed-up time-of-day clock. Provisions are made for the super powerful Motorola MC68881 floating point math co-processor, for heavy math and number crunching applications. An optional network interface uses one serial (four (4) standard, expandable to 20) as a 125/bit per second network channel, Supports as many as 32 nodes.

The MUSTANG-020 is ideally suited to a wide variety of applications. It provides a cost effective alternative to the other MC68020systems now available. It is an excellent introductory tool to the world of hi-power, hi-speed new generation "super micros". In practical applications it has numerous applications, ranging from scientific to education. It is already being used by government agencies, labs, universities, business and practically every other critical applications center, worldwide, where true multi-user, multi-tasking needs exist. The MUSTANG-020 is UNIX C level V compatible. Where low cost and power is a must, the MUSTANG-020 is the answer, as many have discovered. Proving that price is not the standard for quality!

As a software development station, a general purpose scientific or small to medium business computer, or a super efficient real-time controller in process control, the MUSTANG-020 is the cost effective choice. With the optional MC68881 floating point math co-processor installed, it has the capability of systems costing many times over it's total acquisition cost.

With the DATA-COMP "total package", consisting of a heavy duty metal cabinet, switching power supply with rf/line by-passing, 5 inch DS/DD 80 track floppy, Xebec hard disk controller, 25 megabyte winchester hard disk, four serial RS-232 ports and a UNIX C level V compatible multi-tasking, multi-user operating system, the price is under $4300, w/12.5 megahertz system clock (limited time offer). Most all popular high level languages are available at very reasonable cost. The system is expandable to 32serial ports, at a cost ofless than $65 per port, in multiples of 8 port expansion options.

The SBC fully populated, quality tested, with 4 serial ports pre-wired and board mounted is available for less that $2500. Quantity discounts are available for OEM and special applications, in quantity. All that is required to bring to complete "system" standards is a cabinet, power supply, disks and operating system. All these are available as separate items from DATA-COMP.

Available 12.5 - 25 Mhz systems, call for special prices

A special version of the Motorola 020-BUG is installed on each board. 020-BUG is a ROM based debugger package with facilities for downloading and executing user programs from a host system. It includes commands for display and modification of memory, breakpoint capabilities, a powerful assembler/disassemble and numerous system diagnostics. Various 020-BUG system routines, such as I/O handlers are available for user programs.

Normal system speed is 3-4.5 MIPS, with burst up to 10 MIPS, at 16.6 megahertz. Intelligent I/O available for some operating systems.

Hands-on "actual experience sessions", before you buy, are available from DATA-COMP. Call or write for additional information or pricing.

Data-Comp Division

CPI A Decade of Quality Service*
Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

Mustang-020 Mustang-08 Benchmarks

	32 bit Integer	Register Long
IBM AT 7300 Xenix Sys 3	9.7	
AT&T 7300 UNIX PC 68010	7.2	4.3
DEC VAX 11/780 UNIX Berkley 4.2	3.6	3.2
DEC VAX 11/750 " "	5.1	3.2
68008 OS-9 68K 8 Mhz	18.0	9.0
68000 OS-9 68K 10 Mhz	6.5	4.0
MUSTANG-08 68008 OS-9 68K 10 Mhz	*9.8*	*6.3*
MUSTANG-020 68020 OS-9 68K 16 Mhz	*2.2*	*0.88*
MUSTANG-020 68020 MC68881 UniFLEX 16 Mhz	*1.8*	*1.22*

```
Main()
            {
                register long i;
                for (i=0; i < 999999; ++i);
            }
```

Estimated MIPS - MUSTANG-020 ..... 4.5 MIPS,

Burst to 8 - 10 MIPS: Motorola Specs

Mustang Specifications

12.5 Mhz (optional 16.6 Mhz available) MC68020 full 32-bit wide path
32-bit wide data and address buses, non-multiplexed
on chip instruction cache
object code compatible with all 68XXX family processors
enhanced instruction set - math co-processor interface
68881 math hi-speed floating point co-processor (optional)
direct extension of full 68020 instruction set
full support IEEE P754, draft 10.0
transcendental and other scientific math functions
2 Megabyte of SIP RAM (512 x 32 bit organization)
up to 256K bytes of EPROM (64 x 32 bits)
4 Asynchronous serial I/O ports standard
optional to 20 serial ports
standard RS-232 interface
optional network interface
Yered 8 bit parallel port (1/2 MC68230)
Centronics type pinout
expansion connector for I/O devices
16 bit data path
256 byte address space
2 interrupt inputs
clock and control signals
Motorola I/O Channel Modules
time of day clock/calendar w/battery backup
controller for 2, 5 1/4" floppy disk drives
single or double side, single or double density
35 to 80 track selectable (48-96 TPI)
SASI interface
programmable periodic interrupt generator
interrupt rate from micro-seconds to seconds
highly accurate time base (5 PPM)
5 bit sense switch, readable by the CPU
Hardware single-step capability

CPI™
Mustang Hi-Speed Systems
Only from Data-Comp Div.
68008-68030

Don't be mislead! ONLY Data-Comp delivers the Super MUSTANG-020

OS-9

OS-9 Professional Ver	$850.00
*Includes C Compiler	
Basic09	300.00
C Compiler	500.00
68000 Disassembler (w/source add: $100.00)	100.00
Fortran 77	750.00
Microware Pascal	500.00
Omegasoft Pascal	900.00
Stylo-Graph	495.00
Stylo-Spell	195.00
Stylo-Merge	175.00
Stylo-Graph-Spell-Merge	695.00
PAT w/C source	229.00
JUST w/C source	79.95
PAT/JUST Combo	249.50
Sculptor+ (see below)	995.00
COM	125.00

UniFLEX

UniFLEX (68020 ver)	$450.00
Screen Editor	150.00
Sort-Merge	200.00
BASIC/PreCompiler	300.00
C Compiler	350.00
COBOL	750.00
CMODEM w/source	100.00
TMODEM w/source	100.00
X-TALK (see Ad)	99.95
Cross Assembler	50.00
Fortran 77	450.00
Sculptor+ (see below)	995.00

Standard MUSTANG-020™ shipped 12.5 Mhz.	
Add for 16.6 Mhz 68020	375.00
Add for 16.6 Mhz 68881	375.00
Add for 20 Mhz 68020/RAM	750.00
16 Port exp. RS-232	335.00
Requires 1 or 2 Adapter Cards below.RS232 Adapter	165.00

Each card supports 4 additional ser. ports
(total of 36 serial ports supported)

60 line Parallel I/O card	398.00

Uses 3 68230 Interface/Timer chips,
6 groups of 8 lines each, separate buffer
direction control for each group.

Prototype Board	75.00

areas for both dip and PGA devices & a
pre-wired memory area up to 512K DRAM.

SBC-AN	475.00

Interface between the system and
ARCNET modified token passing LAN, fiber optics optional - call.

LAN software drivers	120.00
Expansion for Motorola I/O Channel Modules	$195.00

Special for complete MUSTANG-020™ system buyers - Sculptor+ $695.00. SAVE $300.00

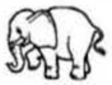
Software Discounts

All MUSTANG-020™ system and board buyers are entitled to discounts on all listed software: 10-70% depending on item. Call or write for quote. Discounts apply after the sale as well.

The PRO!

These hi-speed 68020 systems are presently working at NASA, Atomic Energy Commission, Government Agencies as well as Universities, Business, Labs, and other Critical Applications Centers, worldwide, where speed, math crunching and multi-user, multi-tasking UNIX C level V compatability and low cost is a must.

Only the "PRO" Version of OS-9 Supported!



This is *HEAVY DUTY* Country!

For a limited time we will offer a $400
trade-in on your old 68XXX SBC.
Must be working properly and
complete with all software, cables and
documentation.
Call for more information

Price List:

Mustang-020 SBC		$2490.00
Cabinet w/switching PS		$299.95
5"-80 track floppy DS/DD		$269.95
Floppy Cable		$39.95
OS-9 68K Professional Version		$850.00
C Compiler ($500 Value)		N/C
Winchester Cable		$39.95
Winchester Drive 25 Mbyte		$895.00
Hard Disk Controller		$395.00
Shipping USA UPS		$20.00
UniFLEX	Less	$100.00
MC68881 f/p math processor	Add	$275.00
16.67 Mhz MC68020		$375.00
16.67 Mhz MC68881		$375.00
20 Mhz MC68020 Sys		$750.00
Note all 68881 chips work with 20 Mhz Sys		
	Total:	$5299.80

Save $1000.00
Complete
25 Mbyte HD System
$4299.80
85Mbyte HD System
$5748.80

Note: Only Professional OS-9 Now Available (68020 Version)
Includes ($500) C Compiler - 68020 & 68881 Supported -
For UPGRADES Write or Call for Professional OS-9 Upgrade Kit

Data-Comp Division

CPI A Decade of Quality Service
Systems World-Wide
Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

The C Programmers Reference Source.
Always Right On Target!

C User Notes

A Tutorial Series

By: Dr. E. M. 'Bud' Pass
1454 Latta Lane N.W.
Conyers, GA 30207
404 483-1717/4570
Computer Systems Consultants

INTRODUCTION

This chapter continues the discussion and presentation of a public-domain portable math library written in C by Fred Fish.

MATH LIBRARY

The acos.c function returns the arc cosine of its argument.

```
/*
 *      acos    double precision arc cosine
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

static char     funcname[] = "acos";

double     acos (x)
double     x;
{
    double     y;
    extern double     atan();
    extern double     sqrt();
    auto struct exception xcpt;

    DBUG_ENTER (funcname);
    DBUG_3 ("acosin", "arg %le", x);
    if (x > 1.0 || x < -1.0)
    {
        xcpt.type = DOMAIN;
        xcpt.name = funcname;
        xcpt.arg1 = x;
        if (!matherr (&xcpt))
        {
            fprintf (stderr, "%s: DOMAIN error\n",
funcname);
            errno = EDOM;
            xcpt.retval = 0.0;
        }
    }
    else
    if (!x)
    {
        xcpt.retval = HALFPI;
    }
    else
    if (x == 1.0)
    {
        xcpt.retval = 0.0;
    }
    else
    if (x == -1.0)
    {
        xcpt.retval = PI;
    }
    else
    {
        y = atan ( sqrt (1.0 - (x * x)) / x );
        if (x > 0.0)
        {
            xcpt.retval = y;
        }
        else
        {
            xcpt.retval = y + PI;
        }
    }
    DBUG_3 ("acosout", "result %le", x);
    DBUG_RETURN (x);
}
```

The acosh.c function returns the hyperbolic arc cosine of its argument.

```
/*
 *      acosh   double precision hyperbolic arc cosine
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

static char     funcname[] = "acosh";

double     acosh (x)
double     x;
{
    auto struct exception xcpt;
    extern double     log ();
    extern double     sqrt ();

    DBUG_ENTER (funcname);
    DBUG_3 ("acoshin", "arg %le", x);
    if (x < 1.0)
    {
        xcpt.type = DOMAIN;
        xcpt.name = funcname;
        xcpt.arg1 = x;
        if (!matherr (&xcpt))
        {
            fprintf (stderr, "%s: DOMAIN error\n",
funcname);
            errno = ERANGE;
            xcpt.retval = 0.0;
        }
    }
    else
    if (x > SQRT_MAXDOUBLE)
    {
        xcpt.type = OVERFLOW;
        xcpt.name = funcname;
        xcpt.arg1 = x;
        if (!matherr (&xcpt))
        {
            fprintf (stderr, "%s: OVERFLOW error\n",
funcname);
            errno = ERANGE;
            x = SQRT_MAXDOUBLE;
            xcpt.retval = log (2 * SQRT_MAXDOUBLE);
        }
    }
    else
    {
        xcpt.retval = log (x + sqrt (x * x - 1.0));
    }
    DBUG_3 ("acoshout", "result %le", xcpt.retval);
    DBUG_RETURN (xcpt.retval);
}
```

The asin.c function returns the arc sine of its argument.

```
/*
 *      asin   double precision arc sine
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

static char     funcname[] = "asin";

double    asin (x)
double    x;
{
    extern double    atan ();
    extern double    sqrt ();
    struct exception xcpt;

    DBUG_ENTER (funcname);
    DBUG_3 ("asinin", "arg %le", x);
    if ( x > 1.0 || x < -1.0)
    {
        xcpt.type = DOMAIN;
        xcpt.name = funcname;
        xcpt.arg1 = x;
        if (!matherr (&xcpt))
        {
            fprintf (stderr, "%s: DOMAIN error\n",
funcname);
            errno = EDOM;
            xcpt.retval = 0.0;
        }
    }
    else
    if (!x)
    {
        xcpt.retval = 0.0;
    }
    else
    if (x == 1.0)
    {
        xcpt.retval = HALFPI;
    }
    else
    if (x == -1.0)
    {
        xcpt.retval = -HALFPI;
    }
    else
    {
        xcpt.retval = atan ( x / sqrt (1.0 - (x * x))
);
    }
    DBUG_3 ("asinout", "result %le", xcpt.retval);
    DBUG_RETURN (xcpt.retval);
}
```

The asinh.c function returns the hyperbolic arc sine of its argument.

```
/*
 *      asinh   double precision hyperbolic arc sine
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

static char     funcname[] = "asinh";

double    asinh (x)
double    x;
{
    auto struct exception xcpt;
    extern double    log ();
    extern double    sqrt ();

    DBUG_ENTER (funcname);
    DBUG_3 ("asinhin", "arg %le", x);
    if (x < -SQRT_MAXDOUBLE || x > SQRT_MAXDOUBLE)
    {
        xcpt.type = OVERFLOW;
        xcpt.name = funcname;
        xcpt.arg1 = x;
        if (!matherr (&xcpt))
        {
            fprintf (stderr, "%s: OVERFLOW error\n",
funcname);
            errno = ERANGE;
            xcpt.retval = log (2 * SQRT_MAXDOUBLE);
        }
    }
    else
    {
        xcpt.retval = log (x + sqrt(x * x + 1.0));
    }
    DBUG_3 ("asinhout", "result %le", xcpt.retval);
    DBUG_RETURN (xcpt.retval);
}
```

The atan.c function returns the arc tangent of its argument.

```
/*
 *      atan   double precision arc tangent
 */
```

```
#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

static double    atan_coeffs[] =
{
     .99999999999999999849899, /* P0 must be first
*/
    -.333333333333299308717,
    .1999999999872944792,
    -.142857141028255452,
    .11111097898051048,
    -.0909037114191074,
    .0767936869066,
    -.06483193510303,
    .0443895157187            /* Pn must be last      */
};

static char    funcname[] = "atan";

#define LAST_BOUND 0.2679491924311227074725     /* tan
(PI/12) */

double    atan (x)
double    x;
{
    register int    order;
    double    xt2;
    double    t1;
    double    t2;
    extern double    poly ();
    auto struct exception xcpt;

    DBUG_ENTER (funcname);
    DBUG_3 ("atanin", "arg %le", x);
    if (x < 0.0)
    {
        xcpt.retval = -(atan (-x));
    }
    else
    if (x > 1.0)
    {
        if (x < MAXDOUBLE && x > -MAXDOUBLE)
        {
            xcpt.retval = HALFPI - atan (1.0 / x);
        }
        else
        {
            xcpt.type = UNDERFLOW;
            xcpt.name = funcname;
            xcpt.arg1 = x;
            if (!matherr (&xcpt))
            {
                    fprintf (stderr, "%s: UNDERFLOW
error\n", funcname);
                errno = EDOM;
                xcpt.retval = 0.0;
            }
        }
    }
    else
    if (x > LAST_BOUND)
    {
        t1 = x * SQRT3 - 1.0;
        t2 = SQRT3 + x;
        xcpt.retval = SIXTHPI + atan (t1 / t2);
    }
    else
    if (x < X16_UNDERFLOWS)
    {
        xcpt.type = PLOSS;
        xcpt.name = funcname;
        xcpt.arg1 = x;
        if (!matherr (&xcpt))
        {
            fprintf (stderr, "%s: PLOSS error\n",
funcname);
            errno = EDOM;
            xcpt.retval = x;
        }
    }
    else
    {
        xt2 = x * x;
        order = sizeof (atan_coeffs) / sizeof(double);
        order -= 1;
         xcpt.retval = x * poly (order, atan_coeffs,
xt2);
    }
    DBUG_3 ("atanout", "result %le", xcpt.retval);
    DBUG_RETURN (xcpt.retval);
}
```

The atan2.c function returns the arc tangent of its arguments, equivalent to atan(y / x).

```
/*
 *      atan2   double precision arc tangent of two
arguments
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

double    atan2 (x, y)
double    x;
double    y;
{
    double    result;
    extern double    sign();
    extern double    atan();

    ENTER ("atan2");
    DEBUG4 ("atan2in", "x = %le y = %le", x, y);
    if (!x)
    {
        result = sign (HALFPI, y);
    }
    else
    if (x > 0.0)
    {
        result = atan (y / x);
    }
    else
    {
        result = atan (y / x) + sign (PI, y);
    }
    DEBUG3 ("atan2out", "result %le", result);
    LEAVE ();
    return (result);
}
```

The atanh.c function returns the hyperbolic arc tangent of its argument.

```
/*
 *      atanh   double precision hyperbolic arc tangent
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

static char    funcname[] = "atanh";

double    atanh (x)
double    x;
{
    auto struct exception xcpt;
    extern double    log ();

    DBUG_ENTER (funcname);
    DBUG_3 ("atanhin", "arg %le", x);
    if (x <= -1.0 || x >= 1.0)
    {
        xcpt.type = DOMAIN;
        xcpt.name = funcname;
        xcpt.arg1 = x;
        if (!matherr (&xcpt))
        {
              fprintf (stderr, "%s: DOMAIN error\n",
funcname);
             errno = ERANGE;
             xcpt.retval = 0.0;
        }
    }
    else
    {
        xcpt.retval = 0.5 * log ((1 + x) / (1 - x));
     }
     DBUG_3 ("atanhout", "result %le", xcpt.retval);
     DBUG_RETURN (xcpt.retval);
}
```

The cabs.c function returns the complex absolute value of its argument.

```
/*
 *      cabs   double precision complex absolute value
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

double    cabs (z)
COMPLEX z;
{
    double    result;
    extern double    sqrt ();

    ENTER ("cabs");
      DEBUG4  ("cabsin", "arg %le +j %le", z.real,
z.imag);
      result = sqrt ((z.real * z.real) + (z.imag *
z.imag));
     DEBUG3 ("cabsout", "result %le", result);
     LEAVE ();
     return (result);
}
```

The cacos.c function returns the complex arc cosine of its argument.

```
/*
 *      cacos   complex double precision arc cosine
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX cacos (z)
COMPLEX z;
{
    COMPLEX temp;
    double    swaptemp;
    extern COMPLEX cmult (), csqrt (), clog ();

    ENTER ("cacos");
    DEBUG4 ("cacosin", "arg %le %le", z.real, z.imag);
    temp = cmult(z, z);
    temp.real = 1.0 - temp.real;
    temp.imag = -temp.imag;
    temp = csqrt (temp);
    swaptemp = temp.real;
    temp.real = -temp.imag;
    temp.imag = swaptemp;
    temp.real += z.real;
    temp.imag += z.imag;
    temp = clog (temp);
    z.real = temp.imag;
    z.imag = -temp.real;
     DEBUG4 ("cacosout", "result %le %le", z.real,
z.imag);
    LEAVE ();
    return (z);
}
```

The cadd.c function returns the complex sum of its arguments.

```
/*
 *      cadd   double precision complex addition
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX cadd (z1, z2)
COMPLEX z1;
COMPLEX z2;
{
    ENTER ("cadd");
    z1.real += z2.real;
    z1.imag += z2.imag;
    LEAVE ();
    return (z1);
}
```

The casin.c function returns the complex arc sine of its argument.

```
/*
 *      casin   complex double precision arc sine
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX casin (z)
COMPLEX z;
{
    COMPLEX temp;
    extern COMPLEX csqrt (), clog (), cmult ();

    ENTER ("casin");
    DEBUG4 ("casinin", "arg %le %le", z.real, z.imag);
    temp = cmult (z, z);
    temp.real = 1.0 - temp.real;
    temp.imag = -temp.imag;
    temp = csqrt (temp);
    temp.real -= z.imag;
    temp.imag += z.real;
    temp = clog (temp);
    z.real = temp.imag;
    z.imag = -temp.real;
     DEBUG4 ("casinout", "result %le %le", z.real,
z.imag);
    LEAVE ();
    return (z);
}
```

The catan.c function returns the complex arc tangent of its argument.

```
/*
 *      catan   complex double precision arc tangent
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX catan (z)
COMPLEX z;
{
    COMPLEX temp;
    double    swaptemp;
    extern COMPLEX cdiv (), clog ();

    ENTER ("catan");
    DEBUG4 ("catanin", "arg %le %le", z.real, z.imag);
    temp.real = -z.real;
    temp.imag = 1.0 - z.imag;
    z.imag += 1.0;
    z = cdiv (z, temp);
    z = clog (z);
    swaptemp = z.real;
    z.real = -0.5 * z.imag;
    z.imag =  0.5 * swaptemp;
     DEBUG4 ("catanout", "result %le %le", z.real,
z.imag);
    LEAVE ();
    return (z);
}
```

The ccos.c function returns the complex cosine of its argument.

```
/*
 *      ccos   complex double precision cosine
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX ccos (z)
COMPLEX z;
{
    COMPLEX result;
    extern double    sin(), cos(), sinh(), cosh();

    ENTER ("ccos");
    DEBUG4 ("ccosin", "arg %le %le", z.real, z.imag);
    result.real = cos(z.real) * cosh(z.imag);
    result.imag = -sin(z.real) * sinh(z.imag);
    DEBUG4 ("ccosout", "result %le %le", result.real,
result.lmag);
    LEAVE ();
    return (result);
}
```

The ccosh.c function returns the complex hyperbolic cosine of its argument.

```
/*
 *      ccosh   complex double precision hyperbolic
cosine
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

COMPLEX ccosh (z)
COMPLEX z;
{
    COMPLEX cexpmz;
    extern COMPLEX cexp ();

    ENTER ("ccosh");
    DEBUG4 ("ccoshin", "arg %le %le", z.real, z.imag);
    cexpmz.real = -z.real;
    cexpmz.imag = -z.imag;
    cexpmz = cexp (cexpmz);
    z = cexp (z);
    z.real += cexpmz.real;
    z.imag += cexpmz.imag;
    z.real *= 0.5;
    z.imag *= 0.5;
     DEBUG4 ("ccoshout", "result %le %le", z.real,
z.imag);
    LEAVE ();
    return (z);
}
```

The cdiv.c function returns the complex quotient of its argument.

```
/*
 *      cdiv   double precision complex division
 */

#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

CDMPLEX cdiv (znum, zden)
COMPLEX znum;
COMPLEX zden;
{
    COMPLEX result;
    double    denom;

    ENTER ("cdiv");
     DEBUG4 ("cdivin", "arg1 %le %le", znum.real,
znum.imag);
     DEBUG4 ("cdivln", "arg2 %le %le", zden.real,
zden.imag);
    denom = (zden.real * zden.real) + (zden.imag *
zden.imag);
    if (!denom)
    {
        pmlerr (C_DIV_ZERO);
        result.real = MAX_POS_DBLF;
        result.imag = 0.0;
    }
    else
    {
        result.real = ((znum.real * zden.real) +
            (znum.imag * zden.imag)) / denom;
        result.imag = ((zden.real * znum.imag) -
            (znum.real * zden.imag)) / denom;
    }
    DEBUG4 ("cdivout", "result %le %le", result.real,
result.imag);
    LEAVE ();
    return (result);
}
```

C PROBLEM

There is no concise answer to the problem of certifying a math library. The functions in the library are generally not very homogeneous, so they may not easily be classified into just a few groups, each of which could then be tested as a unit.

For each function, a series of valid and invalid argument values may be provided, and the results of each argument may be checked against a list of results computed by a previously-certified math library. Although this is really "probing", not "testing" or "certifying", and does not necessarily detect all types of problems, it is usually effective enough to provide some indication that the math library functions are performing correctly.

EXAMPLE C PROGRAM

Following is this month's example C program: it tests the default error-handling capabilities of the portable math library and provides examples of calling many of the functions in the portable math library.

```
/*
 *      testerrors.c   error handler test for portable
math library
 *      tests functions of the optional error handler
for the portable
 *       math library.  tests the COUNT, LOG, and
CONTINUE bits for
 *      each function, along with the task error limit.
 */


#include <stdio.h>
#include "pmluser.h"
#include "pml.h"

char    *test1[] =
{
    "\n******  TEST #1  ******",
    "",
    "   Generate all errors",
    "",
    "    (1) LOG bits, COUNT bits, and CONTINUE bits
set.",
     "    (2) Error limit set high enough to avoid
aborting.",
    "",
    "    This test should generate an error message
for each",
    "   error tested, and the final error count should
be printed",
    "   when done.",
    "\n",
    0
};


char    *test2[] =
{
    "\n******  TEST #2  ******",
    "",
    "   Turn off logging.",
    "",
    "   (1) LOG bits reset.",
    "   (2) COUNT bits and CONTINUE bits set.",
     "    (3) Error limit set high enough to avoid
aborting.",
    "",
    "    This test should not log any error messages
but the error",
    "   count should be the same as test 1.",
    "\n",
    0
};


char    *test3[] =
{
    "\n******  TEST #3  ******",
    "",
    "    Turn off error counting",
    "",
    "    (1) LOG and COUNT bits reset.",
    "    (2) CONTINUE bits set.",
    "    (3) Error limit set to 0.",
    "",
    "    This test should not log or count any errors.
The final",
    "   error count should be zero.",
    "\n",
    0
};


char    *test4[] =
{
    "\n******  TEST #4  ******",
    "",
    "    Enable error limit and abort.",
    "",
     "    (1) LOG bits, COUNT bits, and CONTINUE bits
set.",
    "    (2) Error limit set to 5.",
    "",
     "    This test should abort after logging the
sixth error.",
    "\n",
    0
};



main()
{
    initialize();
    pmllim(40);
    prtdoc(test1);
    gen_errors();
    printf("\nTotal errors counted = %d\n", pmlcnt());

    log_off();
    pmllim(40);
    prtdoc(test2);
    gen_errors();
    printf("\nTotal errors counted = %d\n", pmlcnt());
    count_off();
    pmllim(0);
    prtdoc(test3);
    gen_errors();
    printf("\nTotal errors counted = %d\n", pmlcnt());

    initialize();
    pmllim(5);
    prtdoc(test4);
    gen_errors();

    }



    prtdoc(dp)
    register char    **dp;
    {
        while (*dp)
        {
            printf("%s\n", *dp++);
        }
    }
```

```
initialize()
{
    pmlsfs(EXP_OVERFLOW, LOG | COUNT | CONTINUE);
    pmlsfs(EXP_UNDERFLOW, LOG | COUNT | CONTINUE);
    pmlsfs(SCALE_OVERFLOW, LOG | COUNT | CONTINUE);
    pmlsfs(SCALE_UNDERFLOW, LOG | COUNT | CONTINUE);
    pmlsfs(NEG_SQRT, LOG | COUNT | CONTINUE);
    pmlsfs(LOG_OF_ZERO, LOG | COUNT | CONTINUE);
    pmlsfs(LOG_OF_NEGATIVE, LOG | COUNT | CONTINUE);
    pmlsfs(ACOS_BADARG, LOG | COUNT | CONTINUE);
    pmlsfs(ASIN_BADARG, LOG | COUNT | CONTINUE);
    pmlsfs(TAN_OVERFLOW, LOG | COUNT | CONTINUE);
    pmlsfs(COSH_OVERFLOW, LOG | COUNT | CONTINUE);
    pmlsfs(COSH_UNDERFLOW, LOG | COUNT | CONTINUE);
    pmlsfs(SINH_OVERFLOW, LOG | COUNT | CONTINUE);
    pmlsfs(SINH_UNDERFLOW, LOG | COUNT | CONTINUE);
    pmlsfs(ASINH_OVERFLOW, LOG | COUNT | CONTINUE);
    pmlsfs(ACOSH_BAOARG, LOG | COUNT | CONTINUE);
    pmlsfs(ACOSH_OVERFLOW, LOG | COUNT | CONTINUE);
    pmlsfs(ATANH_BAOARG, LOG | COUNT | CONTINUE);
    pmlsfs(ATAN_UNDERFLOW, LOG | COUNT | CONTINUE);
    pmlsfs(C_DIV_ZERO, LOG | COUNT | CONTINUE);
    pmlsfs(CRCP_OF_ZERO, LOG | COUNT | CONTINUE);
    pmlsfs(DINT_2BIC, LOG | COUNT | CONTINUE);
}


log_off()
{
    pmlcfs(EXP_OVERFLOW, LOG);
    pmlcfs(EXP_UNDERFLOW, LOG);
    pmlcfs(SCALE_OVERFLOW, LOG);
    pmlcfs(SCALE_UNDERFLOW, LOG);
    pmlcfs(NEG_SQRT, LOG);
    pmlcfs(LOG_OF_ZERO, LOG);
    pmlcfs(LOG_OF_NEGATIVE, LOG);
    pmlcfs(ACOS_BADARG, LOG);
    pmlcfs(ASIN_BAOARG, LOG);
    pmlcfs(TAN_OVERFLOW, LOG);
    pmlcfs(COSH_OVERFLOW, LOG);
    pmlcfs(COSH_UNDERFLOW, LOG);
    pmlcfs(SINH_OVERFLOW, LOG);
    pmlcfs(SINH_UNDERFLOW, LOG);
    pmlcfs(ASINH_OVERFLOW, LOG);
    pmlcfs(ACOSH_BADARG, LOG);
    pmlcfs(ACOSH_OVERFLOW, LOG);
    pmlcfs(ATANH_BADARG, LOG);
    pmlcfs(ATAN_UNDERFLOW, LOG);
    pmlcfs(C_DIV_ZERO, LOG);
    pmlcfs(CRCP_OF_ZERO, LOG);
    pmlcfs(DINT_2BIG, LOG);
}


count_off()
{
    pmlcfs(EXP_OVERFLOW, COUNT);
    pmlcfs(EXP_UNDERFLOW, COUNT);
    pmlcfs(SCALE_OVERFLOW, COUNT);
    pmlcfs(SCALE_UNDERFLOW, COUNT);
    pmlcfs(NEG_SQRT, COUNT);
    pmlcfs(LOG_OF_ZERO, COUNT);
    pmlcfs(LOG_OF_NEGATIVE, COUNT);
    pmlcfs(ACOS_BADARG, COUNT);
    pmlcfs(ASIN_BADARG, COUNT);
    pmlcfs(TAN_OVERFLOW, COUNT);
    pmlcfs(COSH_OVERFLOW, COUNT);
    pmlcfs(COSH_UNDERFLOW, COUNT);
    pmlcfs(SINH_OVERFLOW, COUNT);
    pmlcfs(SINH_UNDERFLOW, COUNT);
    pmlcfs(ASINH_OVERFLOW, COUNT);
    pmlcfs(ACOSH_BADARG, COUNT);
    pmlcfs(ACOSH_OVERFLOW, COUNT);
    pmlcfs(ATANH_BADARG, COUNT);
    pmlcfs(ATAN_UNDERFLOW, COUNT);
    pmlcfs(C_DIV_ZERO, COUNT);
    pmlcfs(CRCP_OF_ZERO, COUNT);
    pmlcfs(DINT_2BIG, COUNT);
}


gen_errors()
{
    complex z1, z2;

    z1.r = 1.0;
    z1.i = 0.0;
    z2.r = 0.0;
    z2.i = 0.0;
    printf("Testing for exp overflow.\n");
    exp(100.0);
    printf("Testing for exp underflow.\n");
    exp(-100.0);
    printf("Testing for scale exponent overflow.\n");
    scale(1.0, 500);
    printf("Testing for scale exponent underflow.\n");
    scale(1.0, -500);
    printf("Testing for sqrt argument < 0.\n");
    sqrt(-1.0);
    printf("Testing for ln of zero.\n");
    ln(0.0);
    printf("Testing for ln argument < 0.\n");
    ln(-1.0);
    printf("Testing for acos argument > 1.0\n");
    acos(2.0);
    printf("Testing for asin argument > 1.0\n");
    asin(-2.0);
    printf("Testing for tan overflow\n");
    tan(HALFPI);
    printf("Testing for cosh overflow\n");
    cosh(LN_MAXPOSDBL + 1.0);
    printf("Testing for cosh underflow\n");
    cosh(LN_MINPOSDBL - 1.0);
    printf("Testing for sinh overflow\n");
    sinh(LN_MAXPOSDBL + 1.0);
    printf("Testing for sinh underflow\n");
    sinh(LN_MINPOSDBL - 1.0);
    printf("Testing for asinh overflow\n");
    asinh(2.0 * SQRT_MPDF);
    printf("Testing for acosh argument < 1.0\n");
    acosh(0.0);
    printf("Testing for acosh overflow\n");
    acosh(2.0 * SQRT_MPDF);
    printf("Testing for atanh argument >= 1.0\n");
    atanh(-1.0);
    printf("Testing for atan underflow\n");
    atan(RECIP_MAX);
    printf("Testing for complex / 0\n");
    cdiv(&z1, &z2);
    printf("Testing for complex 1 / 0\n");
    crcp(&z2);
      printf("Testing for dint with no fractional
part\n");
    dint(MAX_POS_DBLF);
}
```

EOF

Basically OS-9

Dedicated to the serious OS-9 user.
The fastest growing users group world-wide!
6809 - 68020

A Tutorial Series

By: Ron Voigts
2024 Baldwin Court
Glendale Heights, IL

GETTING IT GOING

I am fascinated by how things get started. Everything has its origins. There is the origin of the universe. There is the origin of life. Computer systems have their origins too.

Computer systems have their beginnings. Now I am not talking about the list of things that occur when starting up. These include putting in disk and doing something to cause it to bootup. Then, it loads everything into memory. Rather I am thinking about the initial process that gets the system behaving as you would like to do. It is the operating system you expect.

An earlier system that I worked with centered around a jump vector. · This was conveniently located at an agreed upon place. When the system was initiated, it would place what was called a "warm start jump location" into this slot. If anything went wrong, it would always return here and restart itself. When a program was loaded, a new "warm start jump" would be placed here. Say, I loaded Basic, it would stay in Basic, because the "jump" vector was there. When I exited back to the system, the original "jump vector" would be replaced.

Things have come a long way, since then. OS-9 does a far better job. It is in position independent code and does not have a hard address to jump to. And it is not a system that has only one process running. Processes spawn other processes. Sometimes many are started. But it all goes back to some initial process. That is the topic of this month.

Two items must be briefly examined before we look at the process that starts thing going. One, we must look at is the sequence of events that occur when booting OS-9. Two, we must consider the file containing the vital parameters in booting OS-9. Neither of these will be examined in great details, but they are as a preliminary to our study.

Initially some type of hardware reset occurs. This may be pushing a "reset" button, turning it on or a momentary power failure. What ever occurs, the OS-9 kernel goes into action. First it searches memory for ROM's. Next, it determines the amount of RAM available. Then, necessary modules are loaded into memory. These come from OS9Boot. Finally, the system startup task is began. This is a module called SYSGO, short for SYStem GO. This module is this months focus of attention.

Now we know the sequence of events that set thing started. We next examine the module that contains the startup parameters. It is INIT. It is usually located in ROM with the OS-9 kernel. It has useful items like the upper limit of RAM memory, the initial standard I/O path string, number of entries in the system device table and the bootstrap module name. Two items of interest, it contains are the first module to be executed and the default directory name. The first module is SYSGO and the default is D0 (or maybe H0, it isn't set in stone).

INIT determines the first module to be executed. It is usually SYSGO. By the time that SYSGO is initiated, the default directory is /D0. One of these days I will go deeper into the workings of INIT. Feasibly, these could be changed. As I pointed out earlier, if a hard disk were part of your system, H0 could be the default. SYSGO could also be changed. But I have never seen anything else used. We'll proceed along with these.

SYSGO does a number of things. The things that it accomplishes are:

1. Change execution directory to CMDS.
2. Set initial priority.
3. Execute startup file.
4. Start and maintain a running shell.

These are the most basic things to do. They are accomplished in this order with the SHELL executed last. When SYSGO is finished, you will see the word "Shell" and the familiar prompt. Listing 1 is a bare bones version of SYSGO.

An examination of it will prove to be enlightening.

Going to START, we see that the first thing it does is to set up a signal intercept trap. Register X is loaded with the intercept routine's location. This is at INTRCPT,PCR. A call is made to F$Icpt. This tells OS-9 where the signal handler routine is located. Now whenever SYSGO receives a signal, it will go to INTRCPT. All that is located there is a RTI, return from interrupt. The intercept routine is entered and immediately exited again.

Next the X register is loaded with a pointer to the string "Cmds" and the A register is loaded with $04, used for setting the commands directory. A call is made to I$ChgDir. This causes the current execution directory to become /D0/CMDS. Remember that the default was already /D0, as determined at system boot time. So there is no need to change the default working data directory.

Third, the priority is set. A call is made to F$Id, which returns the process ID in the A register. The B register is loaded with $80, a middle priority value. And the call is made to F$Prior. The importance of this is that all processes FORKed from SYSGO will inherit its priority value.

Now comes the execution of the STARTUP file. Register X is loaded with a pointer to "Shell". Register U is loaded with a pointer to "Startup -p". The "-p" turns off the OS-9 prompting, while "Startup" is being executed. Register D (A/B) is loaded with $0100. The language type here is 6809 Object code. The Y register is loaded with the parameter area length. Here it is the length of "Startup -p", plus a trailing carriage return. A call is made to F$Fork and the file /D0/STARTUP is executed. A call is also made to F$Wait, so that SYSGO waits for the process to finish.

Finally, the a SHELL is started using F$FORK. What is interesting is that should the SHELL die, another one will began. As long as no error occurs, this can go on forever. Try it yourself. Type <ESCAPE> or the <CLEAR><BREAK> combination in the Color Computer. The SHELL will die and another one will start. Should an error occur, execution continues and the error handler is executed. For this version of SYSGO is a jump to the address located at $FFFE. If you examine the Color Computer version, it would appear as:

```
ERROR bra ERROR
```

This is of course an endless loop. If you are running OS-9 on the Color Computer and had it hang up, you may have been in this loop.

The intercept handler is a simple RTI or return from interrupt. I think of this as a shield. Should an interrupt come in, it goes here. Nothing happens. It bounces off and returns with the RTI.

The Color Computer version also has a warm start routine that is copied into the first page of memory beginning at absolute address $0071. The routine is something like this:

```
fcb $55
fdb $0074
nop
clr >$FF03
nop
nop
sta >$FFDF
jmp >$EF0E
```

Much of this are the different flags that are used. The CLR and STA routines turn off the timing interrupts and sets the Color Computer into the all RAM mode. The final JMP is into the kernel. So if an reset occurs, this gets executed.

We can sum up what SYSGO has done for us. It has done all of the high level system initialization. This is setting the commands directory, setting the priority level and executing STARTUP with a SHELL. It starts the first user program, which in our case is the SHELL. And it goes into a wait state, ready to restart the SHELL if necessary.

The important thing to note here is that it does system initialization. This is valuable, because it can be used to alter the environment the system comes up in. Many times in the past we have used the startup procedure to change things. Generally speaking, this is probably the simplest method. But SYSGO can do the same thing and it can do somethings that STARTUP cannot.

Listing 2 shows the same SYSGO of Listing 1 with a few alterations. This listing is more of an example. I would probably not use this one in my system. But it is interesting and can show some things that can be done.

I added a startup message. This can be almost anything desired. Starting at MESSAGE are 3 lines telling:

```
OS-9 SYSTEM LEVEL II BY MICROWARE
CONFIGURED FOR STYLO START UP
DEFAULT WORKING DIRECTORY IS /D1
```

Register X is loaded with the location of the message and register Y is loaded with the size. Loading register A with the standard output path, a call is made to I$Write. Poof! There is the start up message.

The next addition is to set the working directory to /D1. This is done by loading the X register with the location of "/D1" and loading A with the $03 for read and write privilege. A call to made to I$ChgDir. Now the system will default to /D1. This is something that could not be done from the procedure STARTUP. I have heard a number of time from users who have tried adding a line like :

```
CHD/D1
```

to STARTUP. They were discouraged when they learned it would not work. The reason is that when the SHELL dies which was executing the start up procedure, everything it changed is gone too. By putting it in SYSGO the change will last and get passed on to children of the process.

The final change I added to SYSGO is to have it start STYLO. This was done by loading X with the location of "STYLO". Executing F$Fork, STYLO will be up and running. Notice that I did not substitute STYLO for the final start up of the SHELL. Otherwise, we could find ourselves in an endless loop of STYLO executions. In this case, STYLO starts once. After that, SHELLs will begin like usual. So, it would not be necessary to restart the system or use STYLO's feature, PASS, to start a SHELL.

As I said earlier this is only an example. I probably would not use anything like this. First, the startup message could easily be put into STARTUP using ECHO to write the 3 lines to the CRT screen. Coming up into /D1 is not all that useful. I would be changing to another working directory, since I have sub-directories on my disk. (Also, the drive comes on and I have to search for something to stick into it.) And finally, if I really wanted STYLO to come up immediately, I would add the following line to STARTUP.

```
STYLO </TERM >/TERM
```

But there are many reasons you might want to customize SYSGO. Imagine you are a game developer. You might want to have some type of notice printed to the CRT. It would contain the game name, copyright notice, and a maybe some brief instructions. Instead of a SHELL executing, the game could be made to execute. If an escape occurred, the game would re-execute.

I have toyed with the idea of creating a SYSGO for my Level 1 system and the RAM disk I use. The RAM disk could be formatted and directories copied to it from /DO by forking to format and directory copying programs. Just before the final SHELL is executed, directories could be changed to /R/CMDS and /R. When the SHELL comes up, I would be in the RAM disk and ready to go.

What if you have a ROMable system with no disk drives? Maybe the system is a controller for automated process. Initializing directories and processing STARTUP will have to go. Instead of the SHELL, the main control program would be forked to. Even a number of processes could be chained together. When one is finished the next one would start.

The possibilities are almost endless. In many instances altering the startup procedure is sufficient. But many applications require SYSGO to be customized. If you have any urges to do something different, try your hand at customizing SYSGO.

That is it for now. Take care. Until next time, have fun!

LISTING 1

```
                          nam   SYSGO
                           ttl   System's First Process
        ***********************************
        *
        * SYSGO for OS9 Level II
        * Generic Version
        * 7-SEP-87
        *
        ***********************************
        *
        * Function:
         *       System startup procedure. This
           *          version does what is necessary
        *      start a SHELL.
        *
        ***********************************
        *
        * Version 1.0      Original.
        *
        ***********************************
        *
                          ifp1
```

```
00025                               endc
00026
00027    0011                       TYPE        set
PRGRM+OBJCT
00028   0001             REVS       set   1
00029
00030    0000  87CD0075                          mod
PGMEND,NAME,TYPE,REVS,START,MEMSIZE
00031
00032         * stack size
00033 D 0000                       org    0
00034 D 0000                       rmb    $100
00035 D 0100             MEMSIZE   equu   .
00036
00037         * Module name
00038   000D 53797347    NAME      fcs    /SysGo/
00039   0012 02          EDITION   fcb    2
00040
00041         * Commands directory located on /D0
00042   0013             CMDS      equ    *
00043   0013 436D6473              fcc    "Cmds"
00044   0017 0D                    fcb    C$CR
00045
00046         * OS-9 Shell
00047   0018             SHELL     equ    *
00048   0018 5368656C              fcc    "Shell"
00049   001D 0D                    fcb    C$CR
00050
00051         * StartUp procedure name
00052   D01E             STARTUP   equ    *
00053    001E 53746172               fcc    "Startup
-p"
00054   0028 0D                    fcb    C$CR
00055
00056         * Execution entry for SysGo
00057   0029             START     equ    *
00058
00059         * Set up for Intersept handler
00060    0029  308D0044                        leax
INTRCPT,PCR
00061   002D 103F09                os9    F$Icpt
00062
00063         * Change Commands directory
00064   0030 308DFFDF              leax   CMDS,PCR
00065   0034 8604                  lda    #$04
00066   0036 103F86                os9    I$ChgDir
00067
00068         * Get this process' ID
00069   0039 103F0C                os9    FSId
00070
00071         * And use it to set its priority
00072   003C C680                  ldb    #$80
00073   003E 103F0D                os9    F$SPrior
00074
00075         * Using the shell, execute STARTUP
00076   0041 3080FF03    DoStUp    leax   SHELL,PCR
00077    0045  338DFFD5                         leau
STARTUP,PCR
00078   0049 CC0100                ldd    #$100
00079   004C 108E000B              ldy    #$0B
00080   0050 103F03                os9    F$Fork
00081   0053 2518                  bcs    ERROR
00082
00083         * Wait for it to end
00084   0055 103F04                os9    F$Wait
00085
00086         * StartUp a SHELL
00087   0058 308DFFBC    DoShell   leax   SHELL,PCR
00088   005C CC0100                ldd    #$100
00089   005F 108E0000              ldy    #$00
00090   0063 103F03                os9    F$Fork
00091   0066 2505                  bcs    ERROR
00092
00093         * Wait for it to end
00094   0068 103F04                os9    F$Wait
00095
00096         * If no errors occurred, do another
00097   006B 24EB                  bcc    DoShell
00098
00099         * Else we bail out and start over
00100 W 006D 6E9FFFFE  ERROR     jmp    [$FFFE]
00101
00102         * Intercept handler
00103   0071 3B          INTRCPT   rti
00104
00105   0072 C9ED7B                emod
00106
00107   0075             PGMEND    equ    *
00108
00109   0075             end
00110
00111

00000 error(s)
00001 warning(s)
$0075 00117 program bytes generated
$0100 00256 data bytes allocated
$1411 05137 bytes used for symbols
```

LISTING 2

```
00001                              nam    SYSGO
00002                               ttl    System's
First Process
00003
**************************************
00004         *
00005         * SYSGO for OS9 Level II
00006         * For Booting into Stylo
00007         * 18-SEP-87
00008         *
00009
**************************************
00010         *
00011         * Funtion:
00012         * System startup procedure.  This
00013         * version prints a startup message,
00014         * defaults to /D1 and initially
00015         * enters STYLO.
00016         *
00017
***************************************
00018         *
00019         * Version 1.0      Original.
00020         *
00021         * Version 1.1      RDV
00022         * Modified to start STYLO and
00023         * change to working drive to /d1
00024         *
00025
****************************************
00026         *
00027                              ifpl
00032                              endc
00033
00034    0011                      TYPE        set
PRGRM+OBJCT
00035   0001             REVS      set   1
00036
00037    0000  87CD010C                         mod
PGMEND,NAME,TYPE,REVS,START,MEMSIZE
00038
00039         * stack size
00040 D 0000                       org    0
00041 D 0000                       rmb    $100
00042 D 0100             MEMSIZE   equ    .
00043
00044         * Module name
00045   000D 53797347    NAME      fcs    /SysGo/
00046   0012 03          EDITION   fcb    3
00047
00048         * Starting message
```

```
00049   0013             MESSAGE  equ    *
00050    0013  4F532D39               fcc    "OS-9
SYSTEM  LEVEL II BY MICROWARE"
00051   0035 0D0A                 fcb    C$CR,C$LF
00052     0037  434F4E46                       fcc
"CONFIGURED FOR STYLO STARTUP"
00053   0053 0D0A                 fcb    C$CR,C$LF
00054   0055 44454641              fcc    "DEFAULT
WORKING DIRECTORY IS /D1"
00055   0075 0D0A                 fcb    C$CR,C$LF
00056   0064             MSIZE    equ    *-MESSAGE
00057
00058          * Commands directory located on /D0
00059   0077             CMDS     equ    *
00060   0077 436D6473             fcc    "Cmds"
00061   007B 0D                   fcb    C$CR
00062
00063          * Working Drive
00064   007C             WDRIVE   equ    *
00065   007C 2F4431               fcc    "/D1"
00066   007F 0D                   fcb    C$CR
00067
00068          * OS-9 Shell
00069   0080             SHELL    equ    *
00070   0080 5368656C             fcc    "Shell"
00071   0085 0D                   fcb    C$CR
00072
00073          * StartUp procedure name
00074   0086             STARTUP  equ    *
00075   0086 53746172              fcc    "Startup
-p"
00076   0090 0D                  ·fcb    C$CR
00077
00078          * STYLO name
00079   0091             STYLO    equ    *
00080   0091 5374796C             fcc    "Stylo"
00081   0096 0D                   fcb    C$CR
00082
00083          * Execution entry for SysGo
00084   0097             START    equ    *
00085
00086          * Set up for Intercept handler
00087     0097  308D006D    ·                  leax
INTRCPT,PCR
00088   009B 103F09               os9    F$Icpt
00089
00090          * Print message
00091     009E  308DFF71                       leax
>MESSAGE,PCR
00092   00A2 108E0064             ldy    #MSIZE
00093   00A6 8601                 lda    #1
00094   00A8 103F8A               os9    I$Write
00095
00096          * Change Commands directory
00097   00AB 308DFFC8             leax   CMDS,PCR
00098   00AF 8604                 lda    #$04
00099   00B1 103F86               os9    I$ChgDir
00100
00101          * Get this process ID
00102   00B4 103F0C               os9    F$Id
00103
00104          * And use it to set its priority
00105   00B7 C680                 ldb    #$80
00106   00B9 103F0D               os9    F$SPrior
00107
00108          * Usng the SHELL, execute STARTUP
00109   00BC 308DFFC0   DoStUp    leax   SHELL,PCR
00110     00C0  338DFFC2                       leau
STARTUP,PCR
```

```
00111   00C4 CC0100               ldd    #$100
00112   00C7 108E000B             ldy    #$0B
00113   00CB 103F03               os9    F$Fork
00114   00CE 2534                 bcs    ERROR
00115
00116          * Wait for it to end
00117   00D0 103F04               os9    F$Wait
00118
00119          * Set the working drive to /dl
00120     00D3  308DFFA5                       leax
WDRIVE,PCR
00121   00D7 8603                 lda    #$03
00122   00D9 103F86               os9    I$ChgDir
00123
00124          * If no errors occurred, do another
00125   00DC 308DFFB1   DoStylo   leax   STYLO,PCR
00126   00E0 CC0100               ldd    #$100
00127   00E3 108E0000             ldy    #$0
00128   00E7 103F03               os9    F$Fork
00129   00EA 2518                 bcs    ERROR
00130
00131          * Wait for it to end
00132   00EC 103F04               os9    F$Wait
00133
00134          * StartUp a SHELL
00135   00EF 308DFF8D   DoShell   leax   SHELL,PCR
00136   00F3 CC0100               ldd    #$100
00137   00F6 108E0000             ldy    #$00
00138   00FA 103F03               os9    F$Fork
00139   00F0 2505                 bcs    ERROR
00140
00141          * Wait for it to end
00142   00FF 103F04               os9    F$Wait
00143
00144          * If no errors occurred, do another
00145   0102 24EB                 bcc    DoShell
00146
00147          * Else we have an error
00148 W 0104 6E9FFFFE   ERROR     jmp    [$FFFE]
00149
00150          * Intercept handler
00151   0108 3B         INTRCPT   rti
00152
00153   0109 3F3264               emod
00154
00155   010C            PGMEND    equ    *
00156
00157   010C            end
00158
00159

00000 error(s)
00001 warning(s)
$010C 00268 program bytes generated
$0100 00256 data bytes allocated
$145C 05212 bytes used for symbols
```

EOF

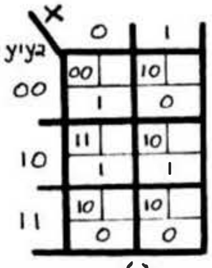
Logically Speaking

The following is the beginning of a continuing series. Most of you will remember Bob from his series of letters on XBASIC. If you like it or want more, let Bob or us know. We want to give you - *what you want!*

The Mathematical Design of Digital Control Circuits

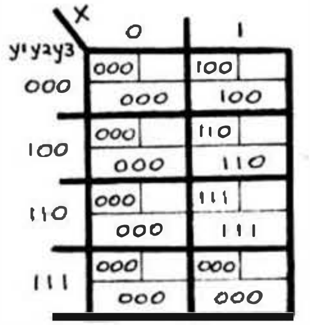
By: R. Jones
Micronics Research Corp.
33383 Lynn Ave., Abbotsford, B.C.
Canada V2S 1E2

SOLUTIONS TO TEST SIX



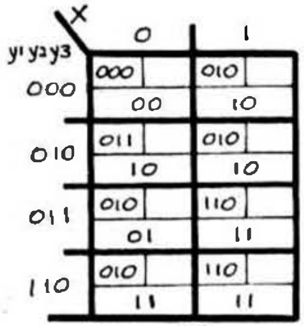
y1y2 \ X	0	1
00	00 1	10 0
10	11 1	10 1
11	10 0	10 0

(1)



y1y2y3 \ X	0	1
000	000 000	100 100
100	000 000	110 110
110	000 000	111 111
111	000 000	000 000

(2)



y1y2y3 \ X	0	1
000	000 00	010 10
010	011 10	010 10
011	010 01	110 11
110	010 11	110 11

(3)

Example 3 is the most complicated circuit-wise, so I'll just take time out to explain its action in the Flow-Table above. Initially we have a stable condition in the top-left square (Location 0-000), with both lights OFF. So it's up to us to get something to happen by pressing push-button X. This moves us to the right into location 1-000, where we see that Light-1 comes ON simultaneously with the coil of Y2 becoming energised. The code 010 in Box-A now makes this location unstable, so circuit-action forces us down to Location 1-010, where the cicuit once more becomes stable, with Light-1 remaining energised. From now on, we'll make every effort to refer to "location" by its more modern equivalent "address", and "code" by its equivalent "instruction", where appropriate. Once again, it's up to us to get something different going, so we release X, which moves us into address 0-010, with Light-1 remaining ON, but Relay Y3's coil becoming energised. However, this address is unstable (why?) and we are forced downwards to address 0-011, where Relay Y3 becomes de-energised and cuts off Light-1, at the same time turning Light-2 ON. This address, too, is unstable, sending us back up to address 0-010, then back down again to address 0-011, and so on, and so on, and .... In this oscillatory condition, Y3 alternately becomes energised and de-energised, causing Lights 1 and 2 to alternate in synchronism. And so it will continue, unless we choose to interrupt it once more!

Let's assume we happen to hit button X just as circuit-action places us in address 0-010. We'll immediately move to the right, and stay there in a stable state with Light-1 ON. So let's release X once more, and return to our previous oscillations. With a little bit of luck we eventually manage to hit X just as the circuit moves into address 0-011, which, of course, pops us to the right into address 1-011, where Relay Y1 now becomes energised for the first time, and also BOTH lights are turned ON. This address turns out to be unstable, and we are pushed down to address 1-110, entering a stable state once more with both lights remaining ON.

If we now release X, we'll move left into address 0-110, where the lights remain ON, but Y1 becomes de-energised, thus setting up the instruction 010. So once more we're unstable, and are compelled to pop up to address 0-010, where Light-2 turns OFF and we find ourselves back in our earlier oscillations.

I hope you didn't find the last Test TOO DIFFICULT. If you'd like me to go into explanations of a few more examples at any time during this series, please write and let me know. Or maybe you can form a study-group with some of your friends. After all, two heads, so they say, are a lot better than one!!

Anyway, if we're all ready and eager to get going, let's move on to Mile 6.

Here's what we've all been looking forward to!!

THE SYNTHESIS OF SEQUENTIAL CONTROL CIRCUITS

Our experiences with Flow-Tables to this point should have made it apparent that the Table accurately describes the ACTION of the circuit undergoing analysis, both in terms of the relays controlling the action and of the output devices being controlled. Although it does not give us a picture of the circuit-network itself, nevertheless it MUST in some way have some sort of relationship to it. After all, we found with ordinary Boolean algebra that we could translate a circuit into a Boolean expression, and we could also reverse the process. Similarly with K-maps.

In the case of sequential circuits, we now know how to translate a circuit diagram into a Flow-Table, and if only we could change this back into a circuit once more we'd feel a lot happier. Well, there is a way to do just that, and this is where our Box-B in each square (unused till now) comes into the picture!

A MORE DETAILED LOOK AT THE FLOW-TABLE

But first, let's think this thing out a little more deeply! We're now in a position to see that Box-A tells us what action the circuit is going to take - - if the code there matches that to the left, then the circuit will remain stable in this position. On the other hand, if the numbers do NOT match, we not only know that the circuit is unstable, but we know PRECISELY where it's going next. It will go to the row called out by the instruction in Box-A. When it gets to this new row, it will "read" and carry out the instruction contained in the new Box-A. In moving around, the outputs (whether they be lights, electric motors, heaters, or anything else) will be forced to change in accordance with the codes contained in Box-C. They themselves, however, play no part in deciding where the circuit action is going next - they simply tag along for the ride.

This means that if we wished we could MAKE the flow-table action move where WE want it to go, simply by inserting the desired instruction in Box-A. If we wished the action to stop at a particular row, all we'd have to do would be to put a number in Box-A which agrees with the row-number, or, if we wished it to move to another row we'd insert instead the number of the desired row. At the same time we'd fill in the Box-Cs with codings describing what we wished to happen to the output devices. Having done all that, we would then complete the Box-Bs in a manner to be described soon, et voila - - provided we haven't made any mathematical errors along the way, here we have a circuit which behaves in exactly the way we want it to. Sounds too simple to be true, doesn't it?

SEQUENTIAL CIRCUIT SYNTHESIS - EXAMPLE ONE

I think the best way to demonstrate the basic technique is to start right at the beginning with a brand-new set of specifications, and take it a step at a time. Our first example will be a very simple one. OK, you guys, which one of you just sighed "Thank goodness for that!"? Anyway, without further ado, let's assume we have a customer who requires a relay control-circuit for a light, L1, which is OFF initially, such that when button X is pressed L1 comes ON and stays ON even when X is released. When X is pressed a second time L1 is to go OFF and stay OFF when X is released. It is to come on again when X is pressed a third time just as it did the first time, then OFF, and so on.

First of all, we draw up a blank flow-table, as shown in Diagram 17a, allowing two columns for the primary control X (one for X=0 and one for X=1) and an uncertain number of rows for the secondary controls, which we number consecutively from 1 onwards. In our example, I numbered from 1 to 5, although, as it turns out, only 4 rows are needed. We use decimal numbers for two reasons - - one being because at this stage of the game we have no idea how many relays we're going to need, and so we can't insert a binary coding as we've been accustomed to do during analysis. The second reason is that our whole design technique is based on using decimal numbers, and this is as good a time to start using them as anywhere else.

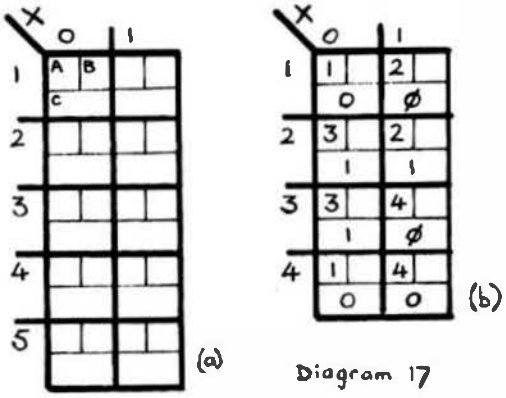


Diagram 17

According to our specs, L1 has to be OFF initially, and, of course, it's understood to be a stable state. Therefore, in the top-left address 0-1 (which is ALWAYS our starting-point), we'll insert a '1' in Box-A to keep the circuit stable and a '0' in Box-C to keep L1 OFF. In case there's still some lingering doubt about the entry in Box-A, remember the rule that in order to make a particular address stable the instruction in Box-A MUST agree with the coding outside the table to the left.

Having disposed nicely of the starting, or switch-on, conditions, let's turn our attention to the next part of the specs, namely that when we press X, L1 has to come ON and stay ON even when we release it. Obviously, pressing X moves us to the right into address 1-1. Now, we COULD insert a 1 in both Boxes A and C, which would take care of the part which says "L1 to come ON when X is pressed", except that when we then release X we'd just slide back to the left into our starting address, and L1 would go out again.

There's no doubt then that we have to move to another row, so we insert the instruction '2' in Box-A (forget about Box-C for the moment) and move downwards to address 1-2, where we insert another 2 in A to keep things stable while we're holding on to button X, and a 1 in C to make L1 come on. Now let's go back to Box-C in the row above, keeping in mind the following three facts :

(a) Phi's are VERY useful to us. The more of them we have, the more scope we have to minimise our network.

(b) A vertical movement in the table indicates that a secondary control has operated, that is, a relay, which means that, time-wise, addresses 1-1 and 1-2 are about 1/100th of a second apart (substantially less if we were using solid-state devices instead).

(c) Although the specs DO say that L1 has to come ON when X is pressed, it is obviously not the intention of our customer to distinguish precisely between the EXACT moment that X is pressed and a moment about 1/100th of a second later.

We'll therefore insert a phi in this spot, and decide later, when developing a minimum circuit, whether to definitely settle for 1 or 0.

Continuing then, and benefitting from our experience with this first movement, we'll code address 0-2 with a 3 in A when we release X, and also in address 0-3, where we come to rest in a stable state. In both addresses we'll code Box-C with a 1 to keep our light ON. So far so good! Our flow-table to date reflects the intention of the specs very nicely up to the end of the first ON-OFF operation of button X.

The rest is fairly simple - - the pattern is exactly the same as that up to this point, except that the condition of L1 is reversed. That is, it's now changing from 1 to 0, instead of from 0 to 1. Observe too that we do not keep the flow-table going on and on, as the specs make it quite clear that at the end of the second ON-OFF operation of X we should be right back at the start of the cycle. We achieve this by coding address 0-4's Box-A with a 1, which shoots us up to the original stable starting-point.

We can now say that our flow-table incorporates the proper sequence of actions called for in the specs, including the correct switching ON or OFF of L1 at the appropriate points in the cycle - apart from the two phi's.

CIRCUIT HAS NOW BEEN DESIGNED

Believe it or not, but we have also designed our circuit. Just as with K-maps, ONCE WE'VE SUCCESSFULLY CODED THE SPECS WE'VE ALSO DESIGNED THE CIRCUIT. All we have to do now (doesn't that sound simple "ALL we have to do ...?) is to read out the circuit diagram from the flow-table. Don't go rushing madly down the trail yet though. We still have a little work to do before this happy state of affairs can result. One thing for sure - even though we haven't the least idea yet as to what the circuit will look like - we can state that we can achieve these effects by using only two relays, which we'll call Y1 and Y2. How can we tell this from the flow-table? Very simply!

Remember that the foundation of our digital system of design (even though we use decimal numbers for convenience) is the binary system of counting. So all we do is to ask ourselves "What power of 2 will cover the number of rows in the table?" 2^1 = 2 and will cover up to 2 rows. 2^2 = 4 and will cover up to 4 rows. 2^3 = 8 and will cover up to 8 rows, and so on. In our example, where we have 4 rows, 2 will have to be raised to the power of 2 (producing 4) in order to cover this number. We'll elaborate on all this in a later example, but before finally committing ourselves to this figure (it COULD possibly turn out to be less) we check the Box-As, row by row, one against the other. For example, in row 1 the Box-As contain the instructions 12; in row 2 they contain 32; in row 3, 34; and in row 4, 14. These number-pairs are all different, and this tells us that we are definitely tied to 4 rows and that we WILL definitely need 2 relays. As I've already mentioned, we'll go into this further in a short while.

THE STATE-DIAGRAM

The first stage in the translation of the flow-table into a circuit-diagram is to construct what is known as a "state diagram", which is shown in Diagram 18.

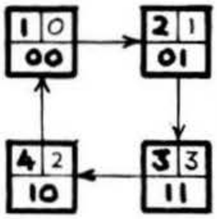


Diagram 18

It consists of a set of boxes, identical with those of the flow-table, separated from each other by about 1/2 inch. In Box-A of each major box we insert the state-numbers through which the machine cycles (in our case 1 - 4), and connect the major boxes together by arrows to duplicate the action-flow in the flow-table. In our example, we commence in State 1, then (as the flow-table shows) we move to State 2, from there to State 3, then to State 4, and finally back to State 1 again.

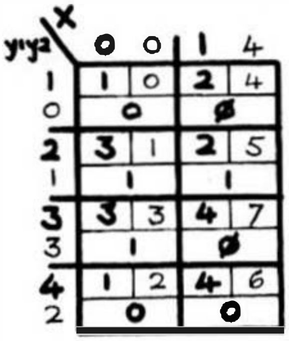
Now, in Box-C of each State-box, we're going to insert a 2-bit binary code (1 bit for each relay in our circuit). The coding will be such that if any two boxes are connected by an arrow the code will differ in ONE bit-position only. In other words, we're going to Gray-code the State-diagram. For now, we'll accept this as a fact-of-life, but later on we'll look at Gray-codes in more detail in order to understand why they're absolutely essential to reliable circuit design. By convention, we always code Box-1 with an all-zero code.

When all the boxes have been successfully coded, we insert in Box-B of the State-diagram the decimal equivalent of these binary numbers, preferably in a contrasting colour. I mentioned a long way back on our journey that I use "red", but will distinguish them with a lighter impression than the instructions in Box-A, even though I shall refer to them as "red-5", "red-19", etc.

We now have a total of three controls in this circuit - one primary control X and two secondary controls Y1 and Y2, which we set out as shown below, with headings according to the binary numbering system. The primary controls are ALWAYS set out first, and then the secondary controls to the right. Thus :

```
4   2   1
X   Y1  Y2
```

We can now add a heading to our flow-table rows, namely y1y2, and in the appropriate rows the red numbers from our State-diagram.



y'y2 \ X	0	0	1	4
1	1	0	2	4
0	0		Ø	
2	3	1	2	5
1	1		1	
3	3	3	4	7
3	1		Ø	
4	1	2	4	6
2	0		0	

Diagram 19

Row 1, according to Box-1 of the State-diagram, corresponds to a red-0, row 2 to a red-1, row 3 to a red-3, and row 4 to a red-2. You'll see that a few lines earlier on, we allocated the bit-value 4 to X, so we add a red-0 to column 0, and a red-4 to column 1 (where X = 1). To round off the completion of the flow-table, we finally fill in all the Box-Bs in the table with a red number corresponding to the sum of its co-ordinates. That is, address 1-2's Box-B has a red-5 inserted because it is also located in Column red-4 and row red-1, and 4 + 1 = 5, and so on for the other addresses.

It's actually the red numbers scattered throughout the flow-table which contain the circuit-diagram, and these we'll read out according to different sets of rules, depending on whether our circuit is going to be composed of relays or solid-state (with slightly different rules if we're going to use an all-NOR implementation, or an all-NAND, and so on - but that's quite a way down the road yet!)

A STUDY OF FLOW-TABLE DATA

Before we actually do the decoding, however, let's take a final look at the flow-table to see what a WEALTH of information it now contains. For instance, Box-A not only tells us whether the circuit is stable or not at this point, but if it's unstable it tells us exactly where the action is going to move. IN ADDITION, it gives us the state of magnetisation of ALL the relay coils in our circuit. Let's for a moment pick out address 0-2. Box-A tells us (a) that the circuit is unstable here, (b) that it's going to move next to row 3, and (c) because code-3 refers us to Box-3 of our state-diagram, we can find out that current is flowing through the coil of both Y1 and Y2, this information being available in both decimal and binary.

Box-B of address 0-2 gives the state of ALL the controls (NOT the coils themselves). In this case the red-1 (binary 001) tells us that X is NOT operated, that Y1's CONTACTS have not yet operated, but Y2's have. Comparing this number with that in Box-A, we can immediately see why the circuit is unstable, in that Y1's coil is energised but its contacts have not yet operated, and we know then that the next move is for them to close in about 1/100th of a second, when it actually moves to row 3. Added to all this data, Box-C tells us that L1 is ON at this moment.

Don't confuse the boxes in the state-diagram with those in the flow-table. They are quite different! The only thing they have in common is Box-A. Box-B in the flow-table gives the state of all the control contacts, while that in the state-diagram gives, in decimal code, the state of energisation of all relay-coils (which is not necessarily the same as that of their contacts). Box-C in the flow-table gives the state of all output devices, while that of the state-diagram merely gives the binary equivalent of its corresponding Box-B.

One final word, and that is that ANY circuit cycling in the sequence shown in our state-diagram, irrespective of the outputs involved, WILL HAVE THE SAME CONTROL CIRCUIT for its relays. We are free to put any pattern of output, or outputs, in Box-C, without affecting Box-A or Box-B, which contain the key to the control-circuit for the relay network.

NOW TO ACTUALLY DECODE THE FLOW-TABLE

The THEORY of the decoding system is too complex to permit of explanation in this tutorial, but the basis of the system used here may be found in the book "Digital Computer and Control Engineering" by R. S. Ledley, published by McGraw-Hill, 1960, pages 336 to 343. This is an excellent book, which covers a wide-ranging field, and which was instrumental in converting me to the decimal system of design which I am here developing for your benefit.

Oh-oh! I think I've already used up my allocation of space in 68MJ, so I think we'll just have to recuperate here for a while. And guess what, you lucky readers? We haven't covered enough new ground for me to give you a meaningful test!! So you get a whole month with nothing to do but wait for details on how to do the circuit-decoding. Hope you can hang on till then.

..... End of Mile 5, ready and eager for Mile 6.

EOF

Corrections: July '87 - Pg. #43, expression at bottom of page should read:
40 K% = ASC(MID$(N$,I%,1)-48+7*(K%>9))

Next line down: should read - "I use +7.....to subtract 7.....

Pg. #47, "UNDERRFLOW" should read "UNDERFLOW"

August '87- Pg. # para 3, line 3: $\underline{AB}$ + $\underline{C}$ should read $\underline{AB+C}$
5: $\underline{5}$ OR $\underline{B}$ OR $\underline{C}$ should read $\underline{A}$OR $\underline{B}$AND $\underline{C}$

August '87
Page 41 - fig. (ii), y4' should have a diagonal bar to indicate "variable".

September '87
Page 23 - Rule 3 Series Circuit, 0.a=0 not =a.

October '87
Page 40 - 2(a) should read "Y" not "Y1"
Page 40 - 3(e) second (e) should read (f).
Page 42 - "b'" should read -0-- not -1--

O.K. folks, now thumb back to page #38: the switch shown should indicated that the "*throw*" should indicate that both elements *throw* on toggle.
TEST ONE: 2(ii) - the output circle should be marked Y_1
Now slide your eyes up near the top of the same page, A1 = y +y2 should read A1=y1 +y2
Also please ignore any "*" included in any of the other listings.

Sorry 'bout all that - but you know - *learning curve!* DMW

Pascal A Tutorial

By: Robert D. Reimiller
Certified Software Corp
616 Camino Caballo
Nipomo, CA 93444
805 929-1359

I missed another month! With a trade show to attend in Germany, and intense work on the 68020 Pascal, it's sometimes hard to keep up with a monthly magazine deadline. The October deadline for the December issue came and went before I realized it.

One of the most important abilities of OS-9 is multi-tasking. In this chapter we will look at two different ways for doing multi-tasking from Pascal.

The first method is to generate a Pascal program for each task and link it into an OS-9 memory module. The Main task would then "Fork" the other tasks that are required. Data can be passed through Pipes or through data modules. Synchronization of tasks can be done using events or signals.

In many cases this method is perfectly satisfactory, especially where each task is a sizable program in itself. Using the 68000 Compiler each program would require a copy of whatever runtime library routines it might need (such as floating point or I/O support), which could add a significant overhead to each program. This option becomes more appealing when the 68020 compiler is released, since it has the option of using traps to access runtime routines, thus the ability to share a common runtime library. The only problem with this method is that using traps to call library routines results in a speed penalty.

Many multi-tasking applications require many small tasks, possibly dealing with diverse I/O ports. For these types of applications, there is a method of having two or more tasks within the same Pascal program.

The method used is an extension called "TASK" procedures. Note that these task procedures are not specific to OS-9, you can use any multi-tasking kernel. The OS-9 package includes interface routines for OS-9, but these can be re-written for just about any multi-tasking environment.

As an example of this method, we will look at an excerpt of code from the "Pascal Shell". This is a menu driven program, described earlier, that keeps track of a particular Pascal project for you. One of the features of the Pascal Shell is the ability to select one or more files for compilation or assembly and have those done in background, thus allowing you to edit another file in foreground.

There is a common data structure that must be accessed by the foreground environment and the background, this is the linked list of file names. This linked list has the structure :

```
type
  file_type = (pascal, assembly, include) ;
  files = record
            link : ^files ;
            options : integer ;
          class : file_type ;
          modified : boolean ;
          name : string[40]
           end ;
  optlist = record
                                              ink
: ^optlist ;
                                            ptnum
: integer ;
                                         ptstring
: string
            end ;
var
  fbase : ^files ;
  obase : ^onext ;
```

When the "shell file" is read in, each name is entered into the "files" linked list and marked as to whether it is a pascal source file, an assembly language source file, or an include file (or other type of file). Variable "fbase" has the start of the linked list.

When a background chain is started in the Pascal Shell, it is started as a new task so the Pascal Shell can continue on as before. The background chain first opens the standard input, output, and error paths for the compilations and assemblies to use. The standard input and error paths are opened to the device "/nil", which indicates that they are not used. The output path, which is used by the Assembler and Linker for any sign-on messages is redirected to a file having the name "back_xxxx" where xxxx is the current task ID. This guarantees that multiple users on the system will not be writing to the same file, since the task ID is unique per user.

The background task then scans the file list for files to be compiled or assembled. These are indicated by having the "options" field set non zero. This options field is a unique number corresponding to a set of compilation/assembly command line options selected from a menu by the user. These would indicate for example, if you want to generate code for the debugger, or for linking, and whether or not you want a listing. When it finds a file that has a non-zero "options" field, it searches the options linked list until it finds a match with the "optnum" field. It then uses the string associated with this option for command line processing.

It then calls a procedure called "shell" that handles starting the compiler or assembler, and then waiting for it to finish. This same procedure is also used by the foreground to call the editor or other utility, since a procedure can be shared by more than one task. When the background task runs out files, it simply continues to the end of the task procedure, which automatically terminates the task.

Now lets get to some specifics on task procedures. As far as the compiler is concerned, the only difference between a regular procedure and a task procedure are in the runtime library routines it calls. When inside a task procedure (including nested procedures), the following runtime calls are modified :

Procedure setup Procedure end Nested procedure start Memavail function New, Dispose, Mark, and Release procedures

The procedure setup for a task procedure is responsible for actually starting the task and returning to the caller. The procedure end will terminate the task. The nested procedure start is different from a normal procedure start routine only in stack checking. The memavail function is different because you have the option of using a task's own heap area, or the global heap. Likewise with the New, Dispose, Mark, and Release procedures, they must act either on the local task's heap, or the global heap.

Why the option of two heaps? If you are building structures that are no longer needed once the task terminates, then you would use the local heap to avoid fragmenting the global heap. On the other hand, if you are building structures on the heap that need to remain after the task is terminated, or need to be shared by multiple tasks, then you would want to use the global heap.

There are a number of task procedure parameters that are absolutely required for any given implementation of multi-tasking. These parameters must be the LAST parameters, however user-defined parameters may be defined before the required parameters. Making the required parameters LAST guarantees that they are a fixed offset from the procedure stack mark. Likewise, certain local variables must be defined FIRST for the same reason.

For the implementation under OS-9, the following parameters are required :

1) Task name, must be unique in system, defined as "name : string[32]" and must be set by the caller.

2) Task stack size, stack size to be used by new task, would also include the local task's heap, if enabled. Defined as "stacksize : longhex", must be set by the caller.

3) Task module header buffer, 256 bytes long, normally defined as array [1..256] of byte on the global stack. The address of this buffer is passed by the caller, since it is defined as "header : longhex".

4) Number of I/O paths to copy from the caller task. As best as can be determined, this paramcter, if non-zero, will start with the caller's path 0 (standard input) and copy this path to the new task. For example : IOPATHS = 2 would copy the standard input and standard output to the new task. IOPATHS = 6 would copy paths 0 through 5 from the caller to the new task. What if you only want to copy path number 5 to the new task, but not 0-4, I don't know, ask Microware. Defined as "iopaths : integer".

5) Priority of new task, 1 - 255. On my 68020 system everything seems to have a priority of 128, I gave the background task a priority of 100 and this seems to work ok, only occasionally slowing down the editor in foreground while compiling in background. Defined as "priority : integer".

6) ID of new task. This is not set by the caller, but is set when the new task starts up. This allows the caller (parent) to send signals to the new (child) task by ID number. Defined as "var newid : hex".

7) Tasking error. This is not set by the caller, but is set non-zero by the new task if there is an error, such as not enough memory, heap overflow, etc. Defined as "taskerr : hex".

8) Global/local heap flag. Set by the caller to tell the task which heap to use. True will use Global heap, false

will use local heap. This variable can be modified by the task itself, but be careful that you don'tallocate offofone heap, and de-allocate off of another. Defined as "globalheap : boolean".

The following local variables must be declared first in the task :

1) Local heap pointer. There is normally no need to access this variable by the task explicitly, it is used by the support routines only. Defined as "heapptr : longhex".

2) Module header pointer. This is the address of the local task's module header that is created during task startup. Defined as "modptr : longhex"

3) Parent task ID code. If the child task needs to send a signal to the parent, this is the ID value to use. Defined as "parid : integer".

4) Child task ID code. This is ID value of the new task. Defined as "myid : integer".

Any other local variables needed will follow those 4 above.

How does all this work? First, the support routine builds a module header for the new task using the 256 byte buffer provided. This module header also contains a small amount of code used to start executing the actual task procedure code. The module header also has values such as the stacksize requested copied from the parameter list. After it has built the module header, it calls the OS-9 routine to set in the correct module header checksum and CRC. It then calls a routine called "vermod" which has been previously loaded into memory. Vermod runs in system state and is used to enter the newly created module header into the system module table. It then terminates. This little bit of trickery is needed since the only way to enter a module into the module directory in user state is to load a file from disk.

Now that we have our new task in the module directory, we executea "fork" call to actually start it, using the values such as priority from the parameter list. The code then sets in the return parameters such as "newid" and returns to the calling program. Should a task procedure end, it simply does an "exit" call resulting in being terminated by the operating system and removed from the module directory.
Here is an excerpt from the Pascal shell, showing the basic code involved in the background chain :

```
procedure background (name : string[32] ;
     stacksize : longhex ; header : longhex ;
      iopaths, priority : integer ;
      var newid, taskerr : hex ;
      globalheap : boolean) ; task ;
  var
    heapptr, modptr : longhex ;
    parid, myid : integer ;
    inpath, outpath, auxpath : text ;
    back_fnext : ^files ;
    back_onext : ^optlist ;
    tilt : boolean ;
    lerr : hex ;
  begin
    reset (inpath, '/nil') ;
     rewrite (outpath, backfile) ; {backfile
set
                                           by
caller}
    rewrite (auxpath, '/nil') ;
    back_fnext := fbase ;
    tilt := false ;
    while back_fnext <> nil do
      begin
        if (back_fnext^.options <> 0) and
          (back_fnext^.class = pascal)
          then
            begin
              back_onext := obase ;
              while back_onext^.optnum <>
                    back_fnext^.options do
                               back_onext  :=
back_onext^.link;
              {use back_onext^.optstring for
                command line processing
               module name will be 'pc'}
              tilt := shell (comstr) ;
              {comstr is command line}
            end ;
        back_fnext := back_fnext^.link
      end ;
    { do same for assembly,
      module name = 'ra' }
    { get ready to terminate this task,
      close paths first}
    lerr := i_close (0) ;
    lerr := i_close (1) ;
    lerr := i_close (2)
  end ; {terminate task}
```

When the main program wants to start the background, it executes something like :

```
  err := f_id (mainid, mainpr, maingrp) ;
  backfile := concat('back_', string(mainid))
;
                                  background
(concat('psback',string(mainid)),
         1000,addr(backhead), 0, 100, newid,
         taskerr, true)
```

The shell function is a useful general purpose routine to call a program with command line options. such as : 'pc -os test .'

```
function shell (comstr : string) : boolean ;
  var
```

```
    type_id, err, cerr, dead_id : hex ;
  begin
    type_id := 0 ;
    shell := false ;
    err := f_fork (comstr, 'shell', 0, 3,
                   0, type_id) ;
    if err <> 0
      then
        begin
          f_perr (err, 0) ;
          shell := true
        end
      else
        begin
          repeat
            err := f_wait (dead_id, cerr)
          until type_id = dead_id ;
          if err <> 0
            then
              begin
                f_perr (err, 0) ;
                shell := true
              end
          else if cerr <> 0
            then
              begin
                f_perr (cerr, 0) ;
                shell := true
              end
       end
  end ;
```

You can see from this example that each task can share global memory, but what if we want to access a global variable, but want some control of when that can be done. One very fast and simple way is to use a semaphore. One common need is to control access to a device, such as a CRT screen. This resource can be controlled by using a byte (or shortinteger or shorthex in 68020 Pascal) variable. To indicate that the resource is available, the byte is set to zero. To request access, use the Pascal boolean function TAS. If the resource was available TAS will set the byte negative indicating in-use, and return true indicating you may now use the resource. If the byte was already negative because someone else is using the resource, then TAS will return false. The TAS function uses the 68000 TAS instruction, which is indivisible, guaranteeing proper task synchronization. When you are done with the resource, simply set the flag to zero, which will indicate to another task that it is now available.

As an example, the procedure "waitdisp" will wait until the resource controlled by the byte variable "out" is available.

```
procedure waitdisp ;
  var
    time : longhex ;
    err : hex register ;
  begin
    while not tas (out) do
      begin
        time := 1 ;
        err := f_sleep (time)
      end
  end ;
```

Next month (I hope!) we will look at generating a program that does not need an operating system, something that sets OmegaSoft Pascal apart from most of the compilers available from operating system vendors.

OmegaSoft is a registered trademark of Certified Software Corporation. OS-9 and OS-9/68000 are trademarks of Microware Systems Corporation.
EOF

Telephone: (615) 842-4600 **South East Media** *Telex: 5106006630*

*OS-9, UniFLEX, FLEX, SK*DOS*
SOFTWARE
!!! Please Specify Your Operating System and Disk Size !!!

SCULPTOR

Full OEM & Dealer Discounts Available!

THE SCULPTOR SYSTEM

Sculptor combines a powerful fourth generation language with an efficient database management system. Programmers currently using traditional languages such as Basic and Cobol will be amazed at what Sculptor does to their productivity. With Sculptor you'll find that what used to take a week can be achieved in just a few hours.

AN ESTABLISHED LEADER

Sculptor was developed by professionals who needed a software development tool with capabilities that were not available in the software market. It was launched in 1981 and since then, with feedback from an ever-increasing customer base, Sculptor has been refined and enhanced to become one of the most adaptable, fast, and above all reliable systems on the market today.

SYSTEM INDEPENDENCE

Sculptor is available on many different machines and for most operating systems, including MS DOS, Unix/Xenix and VMS. The extensive list of supported hardware ranges from small personal computers, through multi-user micros up to large minis and mainframes. Sculptor is constantly being ported to new systems.

APPLICATION PORTABILITY

Mobility of software between different environments is one of Sculptor's major advantages. You can develop applications on a stand-alone PC and – without any alterations to the programs – run them on a large multi-user system. For software writers this means that their products can reach a wider marketplace than ever before. It is this system portability, together with high-speed development, that makes Sculptor so appealing to value added resellers, hardware manufacturers and software developers of all kinds.

SPEED AND EFFICIENCY

Sculptor uses a fast and proven indexing technique which provides instant retrieval of data from even the largest of files. Sculptor's fourth generation language is compiled to a compact intermediate code which executes with impressive speed.

INTERNATIONALLY ACCEPTED

By using a simple configuration utility, Sculptor can present information in the language and format that you require. This makes it an ideal product for software development almost anywhere in the world. Australasia, the Americas and Europe – Sculptor is already at work in over 20 countries.

THE PACKAGE

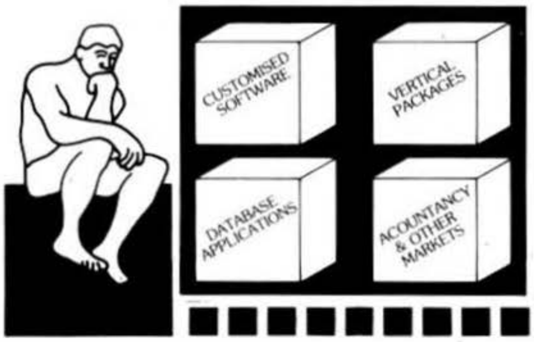
With every development system you receive:

□ A manual that makes sense
□ A periodic newsletter
□ Screen form language
□ Report generator
□ Menu system
□ Query facility
□ Set of utility programs
□ Sample programs

For resale products, the run-time system is available at a nominal cost.

Facts **Features**

Sculptor for 68020
OS-9 & UniFLEX
$995



DATA DICTIONARY

Each file may have one or more record types described. Fields may have a name, heading, type, size, format and validation list. Field type may be chosen from:

□ alphanumeric
□ integer
□ floating point
□ money
□ date

DATA FILE STRUCTURE

□ Packed, fixed length records
□ Money stored in lower currency unit
□ Dates stored as integer day numbers

INDEXING TECHNIQUE

Sculptor maintains a B-tree index for each data file. Program logic allows any numbers of alternative indexes to be coded into one data file.

INPUT DATA VALIDATION

Input data may be validated at three levels:

□ automatic by field type
□ validation list in data dictionary
□ programmer coded logic

ARITHMETIC OPERATORS

-	Unary minus
*	Multiplication
/	Division
%	Remainder
+	Addition
-	Subtraction

RELATIONAL OPERATORS

=	Equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
<>	Not equal to
and	Logical and
or	Logical or
ct	Contains
bw	Begins with

SPECIAL FEATURES

□ Full date arithmetic
□ Echo suppression for passwords
□ Terminal and printer independence
□ Parameter passing to sub-programs
□ User definable date format

MAXIMA AND MINIMA

Minimum key length	1 byte
Maximum key length	160 bytes
Minimum record length	3 bytes
Maximum record length	32767 bytes
Maximum fields per record	32767
Maximum records per file	16 million
Maximum files per program	16
Maximum open files	Operating system limit

PROGRAMS

□ Define record layout
□ Create new indexed file
□ Generate standard screen-form program
□ Generate standard report program
□ Compile screen-form program
□ Compile report program
□ Screen-form program interpreter
□ Report program interpreter
□ Menu interpreter

□ Query facility
□ Reformat file
□ Check file integrity
□ Rebuild index
□ Alter language and date format
□ Setup terminal characteristics
□ Setup printer characteristics

SCREEN FORM LANGUAGE

□ Programmer defined options and logic
□ Multiple files open in one program
□ Default or programmer processing of exception conditions
□ Powerful verbs for input, display and file access
□ Simultaneous display of multiple records
□ Facility to call sub-programs and operating system commands
□ Conditional statements
□ Subroutines
□ Independent of terminal type

MUSTANG-020 Users - Ask For Your Special Discount!

* Tandy CoCo III Special - Reg. $595 * Special $389 *

	*	**	***		*	**	***
MUSTANG-020	$995	$199	$595	PC/XT/AT MSDOS	$595	$119	$595
OS/9 UniFLEX 6809	"	"	"	AT&T 381 UNIX	"	"	"
IBM Compatibles	"	"	"	SWTPC 68010 UniF	$1595	$319	$797
Tandy CoCo III	Special $389.00			SWTPC 68010 UNIX	$1990	$398	$995

... Sculptor Will Run On Over 100 Other Types of Machines ...
... Call for Pricing ...
!!! Please Specify Your Make of Computer and Operating System !!!

* Full Developement Package
** Run Time Only
*** C Key File Library

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343
Telephone: (615) 842-4600 Telex: 5106006630

mastercard

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600 **South East Media** *Telex: 5106006630*

*OS-9, UniFLEX, FLEX, SK*DOS*

DISASSEMBLERS

SUPER SLEUTH from Computer Systems Consultants Interactive Disassembler, extremely *POWERFUL!* Disk File Binary/ASCII Examine/Change, Absolute or FULL Disassembly. XREF Generator, Label "Name Changer", and Files of "Standard Label Names" for different Operating Systems.

Color Computer SS-50 Bus (all w/ A.L. Source)
CCD (32K Req'd) Obj. Only $49.00
F, S, $99.00 - CCF, Obj. Only $50.00 U, $100.00
CCF, w/Source $99.00 O, $101.00
CCO, Obj. Only $50.00
OS9 68K Obj. $100.00 w/Source $200.00

DYNAMITE+ -- Excellent standard "Batch Mode" Disassembler. Includes XREF Generator and "Standard Label" Files. Special OS-9 options w/ OS-9 Version.

CCF, Obj. Only $100.00 - CCO, Obj. $ 59.95
F, S, " " $100.00 - O, object only $150.00
U, " " $300.00

PROGRAMMING LANGUAGES

PL/9 from Windrush Micro Systems -- By Graham Trott. A combination Editor Compiler Debugger. Direct source-to-object compilation delivering fast, compact, re-entrant, ROM-able, PIC. 8 & 16-bit Integers & 6-digit Real numbers for all real-world problems. Direct control over ALL System resources, including interrupts. Comprehensive library support; simple Machine Code interface; step-by-step tracer for instant debugging. 500+ page Manual with tutorial guide.

F, S, CCF - $198.00

PASC from S.E. Media - A FLEX9, SK*DOS Compiler with a definite Pascal "flavor". Anyone with a bit of Pascal experience should be able to begin using PASC to good effect in short order. The PASC package comes complete with three sample programs: ED (a syntax or structure editor), EDITOR (a simple, public domain, screen editor) and CHESS (a simple chess program). The PASC package come complete with source (written in PASC) and documentation.

*FLEX, SK*DOS $95.00*

WHIMSICAL from S.E. MEDIA Now supports *Real Numbers.* "Structured Programming" WITHOUT losing the Speed and Control of Assembly Language! Single-pass Compiler features unified, user-defined I/O; produces ROMable Code; Procedures and Modules (including pre-compiled Modules); many "Types" up to 32 bit Integers, 6-digit Real Numbers, unlimited sized Arrays (vectors only); Interrupt handling; long Variable Names; Variable Initialization; Include directive; Conditional compiling; direct Code insertion; control of the Stack Pointer; etc. Run-Time subroutines inserted as called during compilation. *Normally produces 10% less code than PL/9.*

F, S and CCF - $195.00

KANSAS CITY BASIC from S.E. Media - *Basic for Color Computer OS-9* with many new commands and sub-functions added. A full implementation of the IF-THEN-ELSE logic is included, allowing nesting to 255 levels. Strings are supported and a subset of the usual string functions such as LEFT$, RIGHT$, MID$, STRING$, etc. are included. Variables are dynamically allocated. Also included are additional features such as Peek and Poke. A must for any Color Computer user running OS-9.

CoCo OS-9 $39.95

C Compiler from Windrush Micro Systems by James McCosh. Full C for FLEX, SK*DOS except bit-fields, including an Assembler. *Requires the TSC Relocating Assembler if user desires to implement his own Libraries.*

F, S and CCF - $295.00

C Compiler from Introl -- Full C except Doubles and Bit Fields, streamlined for the 6809. Reliable Compiler; FAST, efficient Code. More UNIX Compatible than most.

*FLEX, SK*DOS, CCF, OS-9 (Level II ONLY), U - $575.00*

PASCAL Compiler from Lucidata -- ISO Based P-Code Compiler. Designed especially for Microcomputer Systems. Allows linkage to Assembler Code for maximum flexibility.

F, S and CCF 5" - $190.00 F, S 8"- $205.00

PASCAL Compiler from OmegaSoft (now Certified Software) -- For the *PROFESSIONAL*; ISO Based, Native Code Compiler. Primarily for Real-Time and Process Control applications. Powerful; Flexible.

OS-9, F, S and CCF - $550.00
OS-9 68000 Version - $900.00

KBASIC - from S.E. MEDIA -- A "Native Code" BASIC Compiler which is now Fully TSC XBASIC compatible. The compiler compiles to Assembly Language Source Code. A NEW, streamlined, Assembler is now included allowing the assembly of LARGE Compiled K-BASIC Programs. Conditional assembly reduces Run-time package.

*FLEX, SK*DOS, CCF, OS-9 Compiler / Assembler $99.00*

CRUNCH COBOL from S.E. MEDIA -- Supports large subset of ANSII Level 1 *COBOL* with many of the useful Level 2 features. Full FLEX, SK*DOS File Structures, including Random Files and the ability to process Keyed Files. Segment and link large programs at runtime, or implemented as a set of overlays. The System requires 56K and CAN be run with a single Disk System. *A very popular product.*

*FLEX, SK*DOS, CCF - $99.95*

FORTH from Stearns Electronics -- A CoCo FORTH Programming Language. Tailored to the CoCo! Supplied on Tape, transferable to disk. Written in FAST ML. Many CoCo functions (Graphics, Sound, etc.). Includes an Editor, Trace, etc. Provides CPU Carry Flag accessibility, Fast Task Multiplexing, Clean Interrupt Handling, etc. for the "Pro". Excellent "Learning" tool!

Color Computer ONLY - $58.95

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

**** Shipping ****
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

FORTHBUILDER is a stand-alone target compiler (crosscompiler) for producing custom Forth systems and application programs. All of the 83-standard defining words and control structures are recognized by FORTHBUILDER.

FORTHBUILDER is designed to behave as much as possible like a resident Forth interpreter/compiler, so that most of the established techniques for writing Forth code can be used without change.

Like compilers for other languages, FORTHBUILDER can operate in "batch mode".

The compiler recognizes and emulates target names defined by CONSTANT or VARIABLE and is readily extended with "compile-time" definitions to emulate specific target words.

FORTHBUILDER is supplied as an executable command file configured for a specific host system and target processor. Object code produced from the accompanying model source code is royalty-free to licensed users.

F, CCF, S - $99.95

DATABASE ACCOUNTING

XDMS from Westchester Applied Business Systems

FOR 6809 FLEX-SK*DOS(5/8")

Up to 32 groups/fields per record! Up to 12 character filed name! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/ Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) oriente which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV

IT'S EASY TO USE!

XDMS-IV keeps data management simple! Rather than design a complex DBMS which hides the true nature of the data, we kept XDMS-IV file oriented. The user view of data relationships is presented in reports and screen output, while the actual data resides in easy to maintain files. This aspect permits customiaed presentation and reports without complex redefinition of the database files and structure. XDMS-IV may be used for a wide range of applications from simple record management systems (addresses, inventory ...) to integrated database systems (order entry, accounting...)

The possibilities are unlimited...

FOR 6809 FLEX-SK*DOS(5/8") **$249.95**

ASSEMBLERS

ASTRUK09 from S.E. Media -- A "Structured Assembler for the 6809" which requires the TSC Macro Assembler.

F, S, CCF - $99.95

Macro Assembler for TSC -- The FLEX, SK*DOS STANDARD Assembler.

Special -- CCF $35.00; F, S $50.00

OSM Extended 6809 Macro Assembler from Lloyd I/O. -- Provides local labels, Motorola S-records, and Intel Hex records; XREF. GeneÔrate OS-9 Memory modules under FLEX, SK*DOS.

*FLEX, SK*DOS, CCF, OS-9 $99.00*

Relocating Assembler/Linking Loader from TSC. -- Use with many of the C and Pascal Compilers.

F, S, CCF $150.00

MACE, by Graham Trott from Windrush Micro Systems -- Co-Resident Editor and Assembler, fast interactive A.I. Programming for small to medium-sized Programs.

F, S, CCF - $75.00

XMACE -- MACE w/Cross Assembler for 6800/1/2/3/8

F, S, CCF - $98.00

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media

5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

**** Shipping ****
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600 **South East Media** Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

UTILITIES

Basic09 XRef from S.E. Media -- This Basic09 Cross Reference Utility is a Basic09 Program which will produce a "pretty printed" listing with each line numbered, followed by a complete cross referenced listing of all variables, external procedures, and line numbers called. Also includes a Program List Utility which outputs a fast "pretty printed" listing with line numbers. Requires Basic09 or RunB.
O & CCO obj. only -- $39.95; w/ Source - $79.95

BTree Routines - Complete set of routines to allow simple implementation of keyed files - *for your programs* - running under Basic09. A real time saver and should be a part of every serious programmers tool-box.
O & CCO obj. only - $89.95

Lucidata PASCAL UTILITIES (Requires Pascal ver 3)

XREF -- produce a Cross Reference Listing of any text; oriented to Pascal Source.

INCLUDE -- Include other Files in a Source Text, including Binary - unlimited nesting.

PROFILER -- provides an Indented, Numbered, "Structogram" of a Pascal Source Text File; view the overall structure of large programs, program integrity, etc. Supplied in Pascal Source Code; requires compilation.
F, S, CCF --- EACH 5" - $40.00, 8" - $50.00

DUB from S.E. Media -- **A UniFLEX BASIC decompiler** Re-Create a Source Listing from UniFLEX Compiled basic Programs. Works w/ ALL Versions of 6809 UniFLEX basic.
U - $219.95

LOW COST PROGRAM KITS from Southeast Media The following kits are available for FLEX, SK*DOS on either 5" or 8" Disk.

1. **BASIC TOOL-CHEST $29.95**
BLISTER.CMD: pretty printer
LINEXREF.BAS: line cross-referencer
REMPAC.BAS, SPCPAC.BAS, COMPAC.BAS:
remove superfluous code
STRIP.BAS: superfluous line-numbers stripper
2. **FLEX, SK*DOS UTILITIES KIT $39.99**
CATS. CMD: alphabetically-sorted directory listing
CATD.CMD: date-sorted directory listing
COPYSORT.CMD: file copy, alphabetically
COPYDATE.CMD: file copy, by date-order
FILEDATE.CMD: change file creation date
INFO.CMD (& INFOGMX.CMD): tells disk attributes & contents
RELINK.CMD (& RELINK82): re-orders fragmented free chain
RESQ.CMD: undeletes (recovers) a deleted file
SECTORS.CMD: show sector order in free chain
XL.CMD: super text lister
3. **ASSEMBLERS/DISASSEMBLERS UTILITIES $39.95**
LINEFEED.CMD: 'modularise' disassembler output
MATH.CMD: decimal, hex, binary, octal conversions & tables
SKIP.CMD: column stripper
4. **WORD - PROCESSOR SUPPORT UTILITIES $49.95**
FULLSTOP.CMD: checks for capitalization
BSTYCIT.BAS (.BAC): Stylo to dot-matrix printer
NECPRINT.CMD: Stylo to dot-matrix printer filter code
5. **UTILITIES FOR INDEXING $49.95**
MENU.BAS: selects required program from list below
INDEX.BAC: word index
PHRASES.BAC: phrase index
CONTENT.BAC: table of contents
INDXSORT.BAC: fast alphabetic sort routine
FORMATER.BAC: produces a 2-column formatted index
APPEND.BAC: append any number of files
CHAR.BIN: line reader

BASIC09TOOLS consist of 21 subroutines for Basic09. 6 were written in C Language and the remainder in assembly. All the routines are compiled down to native machine code which makes them fast and compact.

1. **CFILL** -- fills a string with characters
2. **DPEEK** -- Double peek
3. **DPOKE** -- Double poke
4. **FPOS** -- Current file position
5. **FSIZE** -- File size
6. **FTRIM** -- removes leading spaces from a string
7. **GETPR** -- returns the current process ID
8. **GETOPT** -- gets 32 byte option section
9. **GETUSR** -- gets the user ID
10. **GTIME** -- gets the time
11. **INSERT** -- insert a string into another
12. **LOWER** -- converts a string into lowercase
13. **READY** -- Checks for available input
14. **SETPRIOR** -- changes a process priority
15. **SETUSR** -- changes the user ID
16. **SETOPT** -- set 32 byte option packet
17. **STIME** -- sets the time
18. **SPACE** -- adds spaces to a string
19. **SWAP** -- swaps any two variables
20. **SYSCALL** -- system call
21. **UPPER** -- converts a string to uppercase

For OS-9 - $44.95 - Includes Source Code

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

•• Shipping ••
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600

South East Media

Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

See Review in January 1987 issue of 68 Micro Journal

SOFTOOLS

The following programs are included in object form for immediate application. PL/9 source code available for customization.

READ-ME Complete instructions for initial set-up and operation. Can even be printed out with the included text processor.

CONFIG one time system configuration.

CHANGE changes words, characters, etc. globally to any text type file.

CLEANTXT converts text files to standard FLEX, SK*DOS files.

COMMON compare two text files and reports differences.

COMPARE another check file that reports mis-matched lines.

CONCAT similar to FLEX, SK*DOS append but can also list files to screen.

DOCUMENT for PL/9 source files. Very useful in examining parameter passing aspects of procedures.

ECHO echos to either screen or file.

FIND an improve find command with "pattern" matching and wildcards. Very useful.

HEX dumps files in both hex and ASCII.

INCLUDE a file copy program that will accept "includes" of other disk files.

KWIC allows rotating each word, on each line to the beginning. Very useful in a sort program, etc.

LISTDIR a directory listing program. Not super, but better than CAT.

MEMSORT a high-speed text file sorter. Up to 10 fields may be sorted. Very fast. Very useful.

MULTICOL width of page, number of columns may be specified. A MUST!

PAGE similar to LIST but allows for a page header, page width and depth. Adjust for CRT screen or printer as set up by CONFIG. A very smart print driver. Allows printer control commands.

REMOVE a fast file deleter. Careful, no prompts issued. Zap, and its gone!

SCREEN a screen listing utility. Word wraps text to fit screen. Screen depth may be altered at run time.

SORT a super version of MEMSORT. Ascending/descending order, up to 10 keys, case over-ride, sort on nth word and sort on characters if file is small enough, sorts in RAM. If large file, sort is constrained to size of your largest disk capacity.

TPROC a small but nice text formatter. This is a complete formatter and has functions not found in other formatters.

TRANSLIT sorts a file by x keyfields. Checks for duplications. Up to 10 key files may be used.

UNROTATE used with KWIC this program reads an input file and unfolds it a line at a time. If the file has been sorted each word will be presented in sequence.

WC a word count utility. Can count words, characters or lines.

NOTE: this set of utilities consists of 6 5-1/4" disks or 2 8" disks, w/ source (PL9). 3 5-1/4" disks or 1 8" disk w/o source. Complete set SPECIAL INTRO PRICE:

5-1/4" w/source FLEX - SK*DOS - $129.95
w/o source - $79.95
8" w/source - $79.95 - w/o source $49.95

FULL SCREEN FORMS DISPLAY from Computer Systems Consultants -- TSC Extended BASIC program supports any Serial Terminal with Cursor Control or Memory-Mapped Video Displays; substantially extends the capabilities of the Program Designer by providing a table-driven method of describing and using Full Screen Displays.

F, S and CCF, U - $25.00, w/ Source - $50.00

SOLVE from S.E. Media - OS-9 Levels I and II only. A Symbolic Object/Logic Verification & Examine debugger. Including inline debugging, disassemble and assemble. SOLVE IS THE MOST COMPLETE DEBUGGER we have seen for the 6809 OS-9 series! SOLVE does it all! With a rich selection of monitor, assembler, disassembler, environmental, execution and other miscellaneous commands, SOLVE is the MOST POWERFUL tool-kit item you can own! Yet, SOLVE is simple to use! With complete documentation, a snap! Everyone who has ordered this package has raved! See review - 68 Micro Journal - December 1985. No 'blind' debugging here, full screen displays, rich and complete in information presented. Since review in 68 Micro Journal, this is our fastest mover!

Levels I & II only - OS-9 $69.95

DISK UTILITIES

OS-9 VDisk from S.E. Media -- For Level I only. Use the Extended Memory capability of your SWTPC or Gimix CPU card (or similar format DAT) for FAST Program Compiles, CMD execution, high speed inter-process communications (without pipe buffers), etc. - SAVE that System Memory. Virtual Disk size is variable in 4K increÒments up to 960K. Some Assembly Required.

Level I OS-9 obj. $79.95; w/ Source $149.95

O-F from S.E. Media -- Written in BASIC09 (with Source), includes: REFORMAT, a BASIC09 Program that reformats a chosen amount of an OS-9 disk to FLEX, SK*DOS Format so it can be used normally by FLEX, SK*DOS; and FLEX, a BASIC09 Program that does the actual read or write function to the special O-F Transfer Disk; user-friendly menu driven. Read the FLEX, SK*DOS Directory, Delete FLEX, SK*DOS Files, Copy both directions, etc. FLEX, SK*DOS users use the special disk just like any other FLEX, SK*DOS disk

O - 6809/68000 $79.95

LSORT from S.E. Media - A SORT/MERGE package for OS-9 (Level I & II only). Sorts records with fixed lengths or variable lengths. Allows for either ascending or descending sort. Sorting can be done in either ASCII sequence or alternate collating sequence. Right, left or no justification of data fields available. LSORT includes a full

Availability Legends
O = OS-9 S = SK*DOS
F = FLEX, U = UniFLEX
CC0 = Color Computer OS-9
CCV = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600 **South East Media** *Telex: 5106006630*

*OS-9, UniFLEX, FLEX, SK*DOS*

set of comments and errors messages.
OS-9 $85.00

HIER from S.E. Media - *HIER is a modern hierarchal storage system for users under FLEX, SK*DOS*. It answers the needs of those who have hard disk capabilities on their systems, or many files on one disk - any size. Using HIER a regular (any) FLEX, SK*DOS disk (8 - 5 - hard disk) can have sub directories. By this method the problems of assigning unique names to files is less burdensome. Different files with the exact same name may be on the same disk, as long as they are in different directories. For the winchester user this becomes a must. Sub-directories are the modem day solution that all current large systems use. **Each directory looks to FLEX, SK*DOS like a regular file, except they have the extension '.DIR'.** A full set of directory handling programs are included, making the operation of HIER simple and straightforward. A special install package is included to install HIER to your particular version of FLEX, SK*DOS. Some assembly required. Install indicates each byte or reference change needed. Typically - 6 byte changes in source (furnished) and one assembly of HIER is all that is required. No programming required!
*FLEX - SK*DOS $79.95*

COPYMULT from S.E. Media -- Copy LARGE Disks to several smaller disks. FLEX, SK*DOS utilities allow the backup of ANY size disk to any SMALLER size diskettes (Hard Disk to floppies, 8" to 5", etc.) by simply inserting diskettes as requested by COPYMULT. No fooling with directory deletions, etc. COPYMULT.CMD understands normal "copy" syntax and keeps up with files copied by maintaining directories for both host and receiving disk system. Also includes BACKUP.CMD to download any size "random" type file; RESTORE.CMD to restructure copied "random" files for copying, or recopying back to the host system; and FREELINK.CMD as a "bonus" utility that "relinks" the free chain of floppy or hard disk, eliminating fragmentation.
Completely documented Assembly Language Source files included.
*ALL 4 Programs (FLEX, SK*DOS, 8" or 5") $99.50*

COPYCAT from Lucidata -- *Pascal NOT required.* Allows reading TSC Mini-FLEX, SK*DOS, SSB DOS68, and Digital Research CP/M Disks while operating under SK*DOS , FLEX1.0, FLEX 2.0, or FLEX 9.0 with 6800 or 6809 Systems. COPYCAT will not perform miracles, but, between the program and the manual, you stand a good chance of accomplishing a transfer. Also includes some Utilities to help out. Programs supplied in Modular Source Code (Assembly Language) to help solve unusual problems.
F, S and CCF 5" - $50.00 F, S 8" - $65.00

VIRTUAL TERMINAL from S.E. Media - Allows one terminal to do the work of several. The user may start as many as eight task on one terminal, under *VIRTUAL TERMINAL* and switch back and forth between task at will. No need to exit each one; just jump back and forth. Complete with configuration program. The best way to keep up with those background programs.
O & CCO - obj. only - $49.95

FLEX, SK*DOS DISK UTILITIES from Computer Systems Consultants -- Eight (8) different Assembly Language (w/ Source Code) FLEX, SK*DOS Utilities for every FLEX, SK*DOS Users Toolbox: Copy a File with CRC Errors; Test Disk for errors; Compare two Disks; a fast Disk Backup Program; Edit Disk Sectors; Linearize Free-Chain on the Disk; print Disk Identification; and Sort and Replace the Disk Directory (in sorted order). -- PLUS -- Ten XBASIC Programs including: A BASIC Resequencer with EXTRAs over "RENUM" like check for missing label definitions, processes Disk to Disk instead of in Memory, etc. Other programs Compare, Merge, or Generate Updates between two BASIC Programs, check BASIC Sequence Numbers, compare two unsequenced files, and 5 Programs for establishing a Master Directory of several Disks, and sorting, selecting, updating, and printing paginated listings of these files. A BASIC Cross-Reference Program, written in Assembly Language, which provides an X-Ref Listing of the Variables and Reserved Words in TSC BASIC, XBASIC, and PRECOMPILER BASIC Programs.
ALL Utilities include Source≥ (either BASIC or A.L. Source Code).
F, S and CCF - $50.00
BASIC Utilities ONLY for UniFLEX -- $30.00

COMMUNICATIONS

CMODEM Telecommunications Program from Computer Systems Consultants, Inc. -- Menu-Driven; supports Dumb-Terminal Mode, Upload and Download in non-protocol mode, and the CP/M "Modem7" Christensen protocol mode to enable communication capabilities for almost any requirement. Written in "C".
*FLEX, SK*DOS, CCF, OS-9, UniFLEX, 68000 & 6809h*
Source $100.00 - without Source $50.00

X-TALK from S.E. Media - X-TALK consists of two disks and a special cable, the hookup enables a 6809 SWTPC computer to dump UniFLEX files directly to the UniFLEX MUSTANG-020. This is the ONLY currently available method to transfer SWTPC 6809 UniFLEX files to a 68000 UniFLEX system. Gimix 6809 users may dump a 6809 UniFLEX file to a 6809 UniFLEX five inch disk and it is readable by the MUSTANG-020. The cable is specially prepared with internal connections to match the non-standard SWTPC SO/9 I/O Db25 connectors. A special SWTPC S+ cable set is also available. Users should specify which SWTPC system he/she wishes to communicate with the MUSTANG-020. The X-TALK software is furnished on two disks. One eight inch disk contains S.E. Media modem program C-MODEM (6809) and the other disk is a MUSTANG-020 five inch disk with C-MODEM (68020). Text and binary files may be directly transferred between the two systems. The C-MODEM programs are unaltered and perform as excellent modem programs also. X-TALK can be purchased with or without the special cables, but this special price is available to registered MUSTANG-020 users only.
X-TALK Complete (cable, 2 disks) $99.95

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

**** Shipping ****
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipped Only

***OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.**

Telephone: (615) 842-4600 **South East Media** Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

X-TALK Software (2 disks only) $69.95
X-TALK with CMODEM Source $149.95

XDATA from S.E. Media - A COMMUNICATION Package for the UniFLEX Operating System. Use with CP/M, Main Frames, other UniFLEX Systems, etc. Verifies Transmission using checksum or CRC; Re-Transmits bad blocks, etc.
U - $299.99

EDITORS & WORD PROCESSING

JUST from S.E. Media -- Text Formatter developed by Ron Anderson; for Dot Matrix Printers, provides many unique features. Output "Formatted" Text to the Display. Use the FPRINT.CMD supplied for producing multiple copies of the "Formatted" Text on the Printer INCLUDING IMBEDDED PRINTER COMMANDS (very useful at other times also, and worth the price of the program by itself). "User Configurable" for adapting to other Printers (comes set up for Epson MX-80 with Graftrax); up to ten (10) imbedded "Printer Control Commands". Compensates for a "Double Width" printed line. Includes the normal line width, margin, indent, paragraph, space, vertical skip lines, page length, page numbering, centering, fill, justification, etc. Use with PAT or any other editor.

• Now supplied as a two disk set:
Disk #1: JUST2.CMD object file,
*JUST2.TXT PL9 source:FLEX, SK*DOS - CC*
Disk #2: JUSTSC object and source in C:
*FLEX, SK*DOS - OS9 - CC*

The JTSC and regular JUST C source are two separate programs. JTSC compiles to a version that expects TSC Word Processor type commands, (.pp .sp .ce etc.) Great for your older text files. The C source compiles to a standard syntax JUST.CMD object file. Using JUST syntax (.p .u .y etc.) With all JUST functions plus several additional primer formatting functions. Reference the JUSTSC C source. For those wanting an excellent BUDGET PRICED word processor, with features none of the others have. This is it!
Disk (1) - PL9 FLEX only- F, S & CCF - $49.95
Disk Set (2) - F, S & CCF & OS9 (C version) - $69.95
OS-9 68K000 complete with Source - $79.95

PAT from S.E. Media - A full feature screen oriented TEXT EDITOR with all the best of "PIE™". For those who swore by and loved only PIE, this is for you! All PIE features and much more! Too many features to list. And if you don't like these, change or add your own. PL-9 source furnished. "C" source available soon. Easily configured to your CRT, with special config section.
*Regular FLEX, SK*DOS $129.50*
• SPECIAL INTRODUCTION OFFER • $79.95
SPECIAL PAT/JUST COMBO (w/source)
*FLEX, SK*DOS $99.95*
OS-9 68K Version $229.00
SPECIAL PAT/JUST COMBO 68K $249.00

Note: *JUST in "C" source available for OS-9*

CEDRIC from S.E. Media - A screen oriented TEXT EDITOR with availability of 'MENU' aid. Macro definitions, configurable 'permanent definable MACROS' - all standard features and the fastest 'global' functions in the west. A simple, automatic terminal config program makes this a real 'no hassel' product. Only 6K in size, leaving the average system over 165 sectors for text buffer - appx. 14,000 plus of free memory! Extra fine for programming as well as text.
*FLEX, SK*DOS $69.95*

BAS-EDIT from S.E. Media - A TSC BASIC or XBASIC screen editor. Appended to BASIC or XBASIC, BAS-EDIT is transparent to normal BASIC/XBASIC operation. Allows editing while in BASIC/XBASIC. Supports the following functions: OVERLAY, INSERT and DUP LINE. Make editing BASIC/XBASIC programs SIMPLE! A GREAT time and effort saver. Programmers love it! NO more retyping entires lines, etc. Complete with over 25 different CRT terminal configuration overlays.
*FLEX, CCF, SK*DOS $39.95*

SCREDITOR III from Windrush Micro Systems -- Powerful Screen-Oriented Editor/Word Processor. Almost 50 different commands; over 300 pages of Documentation with Tutorial. Features Multi-Column display and editing, "decimal align" columns (AND add them up automatically), multiple keystroke macros, even/odd page headers and footers, imbedded printer control codes, all justifications, "help" support, store common command series on disk, etc. Use supplied "set-ups", or remap the keyboard to your needs. Except for proportional printing, this package will DO IT ALL!
*6800 or 6809 FLEX, SK*DOS or SSB DOS, OS-9 - $175.00*

SPELLB "Computer Dictionary" from S.E. Media -- OVER 150,000 words! *Look up a word from within your Editor or Word Processor (with the SPH.CMD Utility which operates in the FLEX, SK*DOS UCS).* Or check and update the Text after entry; ADD WORDS to the Dictionary, "Flag" questionable words in the Text, "View a word in context" before changing or ignoring, etc. SPELLB first checks a "Common Word Dictionary", then the normal Dictionary, then a "Personal Word List", and finally, any "Special Word List" you may have specified. SPELLB also allows the use of Small Disk Storage systems.
F, S and CCF - $129.95

STYLO-GRAPH from Great Plains Computer Co. -- A full-screen oriented WORD PROCESSOR -- (uses the 51 x 24 Display Screens on CoCo FLEX/SK*DOS, or PBJ Wordpak). Full screen display and editing; supports the Daisy Wheel proportional printers.

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

•• Shipping ••
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

Telephone: (615) 842-4600 **South East Media** Telex: 5106006630

OS-9, UniFLEX, FLEX, SK*DOS

NEW PRICES 6809 CCF and CCO - $99.95,
F, S or O - $179.95, U - $299.95

STYLO-SPELL from Great Plains Computer Co. -- Fast Computer Dictionary. Complements Stylograph.
NEW PRICES 6809 CCF and CCO - $69.95,
F, S or O - $99.95, U - $149.95

STYLO-MERGE from Great Plains Computer Co. -- Merge Mailing List to "Form" Letters, Print multiple Files, etc., through Stylo.
NEW PRICES 6809 CCF and CCO - $59.95,
F, S or O - $79.95, U - $129.95

STYLO-PAK --- Graph + Spell + Merge Package Deal!!!
F, S or O - $329.95, U - $549.95
O, 68000 $695.00

MISCELLANEOUS

TABULA RASA SPREADSHEET from Computer Systems Consultants -- TABULA RASA is similar to DESKTOP/PLAN; provides use of tabular computation schemes used for analysis of business, sales, and economic conditions. Menu-driven; extensive report-generation capabilities. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

DYNACALC -- Electronic Spread Sheet for the 6809 and 68000.
F, S, OS-9 and SPECIAL CCF - $200.00, U - $395.00
OS-9 68K - $595.00

FULL SCREEN INVENTORY/MRP from Computer Systems Consultants -- Use the Full Screen Inventory System/Materials Requirement Planning for maintaining inventories. Keeps item field file in alphabetical order for easier inquiry. Locate and/or print records matching partial or complete item, description, vendor, or attributes; find backorder or below stock levels. Print-outs in item or vendor order. MRP capability for the maintenance and analysis of Hierarchical assemblies of items in the inventory file. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

FULL SCREEN MAILING LIST from Computer Systems Consultants -- The Full Screen Mailing List System provides a means of maintaining simple mailing lists. Locate all records matching on partial or complete name, city, state, zip, or attributes for Listings or Labels, etc. Requires TSC's Extended BASIC.
F, S and CCF, U - $50.00, w/ Source - $100.00

DIET-TRAC Forecaster from S.E. Media -- An XBASIC program that plans a diet in terms of either calories and percentage of carbohydrates, proteins and fats (C P G%) or grams of Carbohydrate, Protein and Fat food exchanges of each of the six basic food groups (vegetable, bread, meat, skim milk, fruit and fat) for a specific individual. Sex, Age, Height, Present Weight, Frame Size, Activity Level and Basal Metabolic Rate for normal individual are taken into account. Ideal weight and sustaining calories for any weight of the above individual are calculated. Provides number of days and daily calendar after weight goal and calorie plan is determined.
F, S - $59.95, U - $89.95

CROSS ASSEMBLERS

TRUE CROSS ASSEMBLERS from Computer Systems Consultants -- Supports 1802/5, Z-80, 6800/1/2/3/8/11/HC11, 6804, 6805/HC05/146805, 6809/00/01, 6502 family, 8080/5, 8020/1/2/35/C35/39/40/48/C48/49/C49/50/8748/49, 8031/51/8751, and 68000 Systems. Assembler and Listing formats same as target CPU's format. Produces machine independent Motorola S-Text.
*68000 or 6809, FLEX, SK*DOS, CCF, OS-9, UniFLEX*
any object or source each - $50.00
any 3 object or source each - $100.00
Set of ALL object $200.00 - w/source $500.00

XASM Cross Assemblers for FLEX, SK*DOS from S.E. MEDIA -- This set of 6800/1/2/3/5/8, 6301, 6502, 8080/5, and Z80 Cross Assemblers uses the familiar TSC Macro Assembler Command Line and Source Code format, Assembler options, etc., in providing code for the target CPU's.
*Complete set, FLEX, SK*DOS only - $150.00*

CRASMB from LLOYD I/O -- Supports Motorola's, Intel's, Zilog's, and other's CPU syntax for these 8-Bit microprocessors: 6800, 6801, 6303, 6804, 6805, 6809, 6811 (all varieties); 6502, 1802/5, 8048 family, 8051 family, 8080/85, Z8, Z80, and TMS-7000 family. Has MACROS, Local Labels, Label X-REF, Label Length to 30 Chars. Object code formats: Motorola S-Records (text), Intel HEX-Records (text), OS9 (binary), and FLEX, SK*DOS (binary). Written in Assembler ... e.g. Very Fast.

CPU TYPE - Price each:

For:	MOTOROLA	INTEL	OTHER	COMPLETE SET
FLEX9	$150	$150	$150	$399
SK*DOS	$150	$150	$150	$399
OS9/6809	$150	$150	$150	$399
OS9/68K	------	------	------	$432

CRASMB 16.32 from LLOYD I/O -- Supports Motorola's 68000, and has same features as the 8 bit version. OS9/68K Object code Format allows this cross assembler to be used in developing your programs for OS9/68K on your OS9/6809 computer.
*FLEX, SK*DOS, CCF, OS-9/6809 $249.00*

GAMES

RAPIER - 6809 Chess Program from S.E. Media -- Requires FLEX, SK*DOS and Displays on Any Type Terminal. Features: Four levels of play. Swap side. Point scoring system. Two display boards. Change skill level. Solve Checkmate problems in 1-2-3-4 moves. Make move and swap sides. Play white or black. **This is one of the strongest CHESS programs running on any microcomputer,** *estimated USCF Rating 1600+ (better than most 'club' players at higher levels)*
F, S and CCF - $79.95

Availability Legends
O = OS-9, S = SK*DOS
F = FLEX, U = UniFLEX
CCO = Color Computer OS-9
CCF = Color Computer FLEX

VISA

South East Media
5900 Cassandra Smith Rd. - Hixson, Tn. 37343

master card

** Shipping **
Add 2% U.S.A. (min. $2.50)
Foreign Surface Add 5%
Foreign Airmail Add 10%
Or C.O.D. Shipping Only

*OS-9 is a Trademark of Microware and Motorola-*FLEX and UniFLEX are Trademarks of Technical Systems Consultants-*SK*DOS is a Trademark of Star-K Software Systems Corp.

The Macintosh™ Section

Reserved as a

A place for your thoughts

And ours.......

Mac-Watch

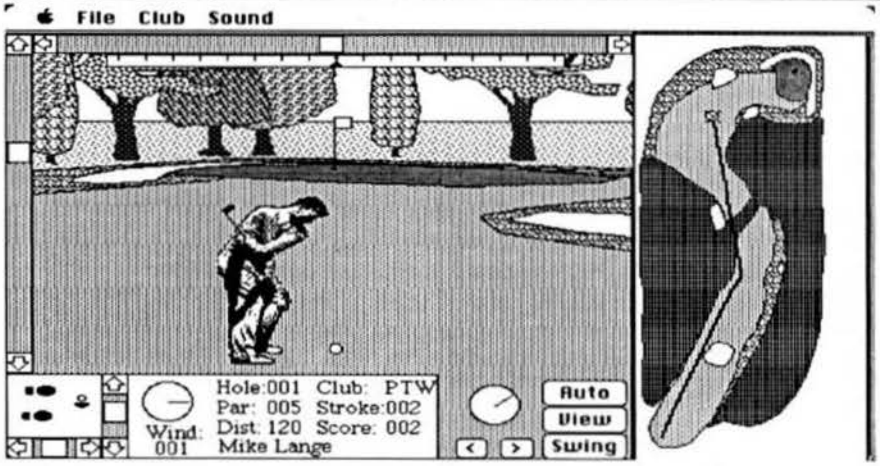
Mac-Golf

By: Gloria Anchors

Until I played MacGolf, the closest I'dcome to the actual game was looking at my mom's clubs in the car trunk. But even though I'd never chased a ball around the golf course, I did enjoy chasing it around the computer screen.

Mac-Golf $59.95
Available from:
Practical Computer Applications

1305 Jefferson HWY
Champlin, Mn 55316
(612) 427-4789
Phone Orders Accepted



Hole:001 Club: PTW
Par: 005 Stroke:002
Dist: 120 Score: 002
Wind: 001
Mike Lange

MacGolf (a game by Practical Computer Applications) has excellent sound and graphics. With each stroke, the ball's flight is depicted in "real time" as it soars, lands, bounces, and rolls to a halt.. I learned to dread the crash, thud, and splash as my misdirected ball went into trees, sandtraps, and water hazards (Out of bounds gets an "Oops!" and a missed put gets "ahs" from the gallery).

The player gets to choose the front or back nine of two specially designed courses, and four others are available from the company. Each well-designed hole is unique and interesting, closely resembling actual course layout with appropriate level of difficulty required. The multi-section screen shows hole layout, the golfer and his point of view, a table with wind speed and direction, distance, par, club, etc. You can choose the clubs, shot direction, strength of shot, and even the position of the golfer's feet. After each stroke, the ball's path is graphically plotted on the map of the hole, and the view window reconstructs itself to show the ball's new position. Seasoned players have commented that the game requires as much concentration for each stroke as the actual outside golf game demands.

If it's fast-paced action you want, MacGolf may not be for you. Like its namesake, play is precise and controlled, requiring skill and practice. Through repeated play you can learn to judge the proper direction and strength for the stroke, and gradually improve your score.

The game's main weakness is the accompanying manual, which provides only sparse information and occasionally complicated instructions. While these instructions are probably adequate for an experienced golfer, they leave the neophyte a little lost. For example, under "Club Selection" the manual says "Each club's distance is not documented in this manual to allow each golfer to develop his/herown individual style and experience." With 14 clubs to choose from in categories like "woods" and "irons", a non-golfer NEEDS the documentation which the manual lacks. (I finally talked to a real golfer about club choice. Only then did I begin to have success with MacGolf.) Also I would like to see some way of taking a mulligan, my husband says they are an essential part of his game and for most all other golfers also.

If you enjoy a challenge, if you've always wanted to play golf but didn't want the exercise, or if you're a real golfer trapped inside by bad weather, try MacGolf. You can experience the frustration and exhilaration of the sport without the long walk to the green.

68XX(X) And The STD BUS

By: Bill West

BILL WEST INCORPORATED
174 Robert Treat Drive
Milford, Connecticut 06460
203 878-9376

Hopefully you are reading this in the Januaty issue of 68 Micro. With the holiday season upon us and all kinds of things to finish before the end of the year, this is being sent out at the last possible minute. Last month (or two months ago), I said I would discuss the Motorola I/O channel to STD interface board we have developed at BWI, including a description of the circuitry used to connect the two different buses. As I start to write this, it seems it will be better to first give a brief description of the I/O channel, and review some products that use it.

The Motorola Remote I/O Channel (RIOC) is intended to allow the connection of inexpensive input and output devices to VME bus or other high performance computer systems. It is a very simple bus structure that includes 12 address lines (A0 - A11), eight bidirectional data lines (D0 - D7), four prioritized interrupt request lines (INT1* - INT4*), a data strobe line (STB*) , a write line (WT*), a data acknowledge line (XACK*), a 4MHz clock (CLK), and a reset line (IORES*). A "*" after a signal name indicates that the signal is active low. A system includes a single bus master and up to sixteen slave devices. The limitation on the numberof slaves is to ensure that the bus drivers are not overloaded.

The I/O channel system bus, or "backplane", is typically a ribbon cable, although it may also be implemented as a regular PC-board type backplane. The cable may be a 50 conductor or 64 conductor cable, and can extend as far as 12 feet. The use of a 64 conductor cable allows the master to supply 5 volt and positive and negative 12 volt power to slave pc boards. If slave subsystems have their own power supplies, the 50 conductor cable carries all the data, address, and control signals. The signals are arranged so that the 50 conductor cable is a subset of the 64 conductor cable, making it relatively easy to interconnect the two types of slaves by splitting a 64 conductor cable into a 50 conductor cable and a 14 conductor cable. Each slave is responsible for decoding and responding to its own I/O address.

Data transfers are accomplished rather simply on the I/O channel. For a read cycle, the master sets up the address lines and then asserts STB*. When the addressed slave has placed the data on the data lines, the slave asserts XACK*. The master deasserts STB* to complete the cycle, followed by the slave deasserting XACK*. A write cycle is similar, except that the master asserts WT* and puts the data to be written on the data lines before asserting STB*, and the slave asserts XACK* when it has latched the data from the data bus. Slaves may interrupt the master by driving one of the interrupt lines low. Each slave which generates interrupts must have a status register the master can read to determine if the slave generated a particular interrupt. The method of clearing an interrupt depends on the design of the particular slave, and may be accomplished by either writing or reading a specified location. Interrupts are normally also cleared whenever the master asserts IORES*, which initializes all the slave devices. The CLK line provides a nominal 4 MHz clock which may be used for timing purposes by the slave devices. Data transfers, however, are asynchronous and do not depend on the CLK signal. For detailed specifications and timing information refer to the I/O Channel Specification Manual, Motorola publication number M68RIOCS/D2.

Motorola provides a number of board level products that can serve as bus masters for the I/O channel. The MVME104 and the MVME110 are two VME bus processor boards that support the I/O channel. The MVME110 is a 68000 based CPU board that includes eight 28-pin sockets for EPROM and static RAM, a 6850 ACIA, and a 6840 counter/timer. The board will function as the system controller in a single CPU system, or can be used in multi-processor systems. Rows A and C of the P2 connector are used for the I/O channel interface. The MVME104 is one of the MVME105 family of CPU boards, and includes a 68010 processor, 512 Kbytes of dual-access dynamic RAM, two 28 pin sockets dedicated to EPROM and two 28 pin sockets that support static RAM or a real-time clock (Dallas Semiconductor type) as well as EPROM. A Zilog

Z8530 is used to provide two serial ports, one of which is configured as an RS232 device and the other as an RS485 device. An MC68230 is used to provide parallel I/O. Four timers are included, of which three are dedicated to hardware support functions including local bus timeout, VME bus timeout, and watchdog timing. The board will function as a system controller, and includes the I/O channel interface on the P2 connector.

The Motorola MVME316 is a dumb interface from the VME bus to the I/O channel. It allows an I/O channel to be added to any VME bus system, and allows direct access from a VME bus master to the devices on the I/O channel. The VIOP is a VME board from Dual Systems Corporation that provides an intelligent interface to the I/O channel. The board includes a 68000 processor, 512 Kbytes of dynamic RAM, and two 28-pin sockets for EPROM. The board can act as a VME bus master or slave, and the I/O channel may be accessed by the on-board 68000 or directly from the VME bus. The board includes circuitry to either generate or respond to VME bus interrupts.

Motorola also provides boards that allow Versabus systems to use I/O channel peripherals. The VM02 is a 68000 based board that includes 128 Kbytes of dynamic RAM, two multi-protocol serial I/O ports, a 6840 counter/timer, and two EPROM sockets, along with the I/O channel interface. The VM03 is a 68010 based board that includes the I/O features of the VM02, with up to 1 megabyte of dynamic RAM, two EPROM sockets, and a 68451 memory management unit, along with an MC146818 real-time clock.

The last system supporting the I/O channel I will discuss is probably the one of most interest to the majority of 68Micro readers. The SBC-BA bus adapter allows the GMX Micro-20 single board computer to use I/O channel peripherals. The Micro-20 is used in a number of systems that have been advertised in 68Micro. The adapter board is a piggy-back board that interfaces the I/O expansion connector on the Micro-20 to the I/O channel, and allows devices on the I/O channel to be directly addressed by the 68020on the Micro-20. The SBC-BA provides the Micro-20 with a full I/O channel interface, with the exception that only two of the interrupt request lines can be supported by the Micro-20 expansion connector.

The above described systems all provide a user with a great deal of computing power. The inclusion of the I/O channel interface seems to provide a means to add a variety of I/O interfaces. Unfortunately, the I/O channel has not been well supported by manufacturers other than Motorola, and even Motorola provides only a limited number of boards for it. As I have mentioned in previous articles, the STD bus provides a wide range of I/O boards, at very reasonable prices. BWI has designed a board, the STD-RIOC, that allows users of systems that support the I/O channel to take advantage of the I/O boards that are available on the STD bus. Up to four STD card cages may be connected to an I/O channel system. The STD-RIOC allows the use of both memory-mapped and I/O-mapped devices on the STD bus, and can support all of the varieties of the STD bus. The interface card provides for the generation of the XACK* signal, and allows STD bus interrupts to be passed over the I/O channel to the main processor.

The fact that the I/O channel can use a ribbon cable to interconnect various subsystems allows system configurations that are not possible with a typical PC-board backplane. Often it is necessary to minimize cable lengths in instrumentation type applications using digital to analog or analog to digital convertors. Other applications simply require many digital control lines that can be difficult to cable back to the main processor. The use of an STD bus I/O subsystem allows the placement of the interface cards in close physical proximity to the devices being monitored or controlled. A single ribbon cable up to 12 feet in length is used to connect the I/O subsystem to the main processor.

Since I want to get this article in the mail, I am going to stop at this point. Next month I plan to discuss the design of the STD-RIOC. Much of the design is implemented with PALs (Programmable Array Logic). If you are unfamiliar with Boolean algebra, you might find it useful to review the recent "Logically Speaking" columns in this magazine by Bob Jones. The easiest and clearest way to implement a design based on PALs is with the kind of tools Bob has been describing.

(PAL is a trademark of Monolithic Memories.)

SOFTWARE USER NOTES

A Tutorial Series

By: Ronald W. Anderson
3540 Sturbridge Court
Ann Arbor, MI 48105

Which Computer

This may come to some of you as a bit of a disappointment, as it does to me. I was sitting the other day contemplating all the computers that I use or have used, and consciously trying to decide which is now my favorite. I guess first of all, I should warn readers that most computer users place availabilityof software high on their list of reasons to purchase a certain brand or compatibility of computer. I am not the "standard" computer user by any stretch ofthe imagination, since I would change hobbies and fields of work if I were forced to use a computer for accounting, inventory, or spreadsheet applications. That leaves only word processing in the present "traditional" computer application areas, and I do weigh the availability of a good text editor and formatter rather highly. In fact I rank these as so important that I have written my own and adapted it to a number of computer configurations, so that more or less becomes a smaller factor also.

Just what then do I look for in determining which is my "favorite" computer? In giving it some thought, I'd have to say that my choice is almost strictly based on performance. I'd rather do about anything than sit and wait for my computer to catch up with me. With that in mind, I'll mention the computers to which I have been exposed lately and then rank them. I have to start out by saying that I have had a 6800 or 6809 based system for over ten years now (hardly seems possible, Don) *(I feel the same way, Ron - DMW)*. I've run Motorola development systems and SWTPc systems, and I've watched others running GIMIX (now GMX) systems. Presently I have access to, and can run an old Tandy Color Computer, an Atari 800 (almost entirely for games), a Tandy 1200-HD, a Tandy 3000 with hard disk, a Mustang 68020 system, and a Peripheral Technology 68008 board with enough hardware around it to make it the equivalent of the Mustang-08 system. First, let's throw out the old Color Computer as having been a toy from the start, though it could have been a very capable system had Tandy chosen to do things just a little bit differently. Then, as I said, the Atari is for games, and I don't have much to go with it, just the disk drive.

Well, as I said, I must base my first choice on which machine I prefer to use to do what I do most. What do I do most? I spend about 70% of my computing time writing and modifying programs in a high level language, in other words editing text files, compiling and debugging programs. Perhaps half of the 70% is directly involved with editing the source files in writing the program. The other half is spent debugging, but about half of that time is spent changing the source file that I have already written. The remaining 30% of my computer time is spent in word processing, also editing text files. I've noticed lately that when I come home, where I have been keeping the Mustang 68020 (which belongs to the company) I'd rather turn that on than the old 6809 system. At work I reach for the 68008 system over the 6809 system. Both those choices are of course contingent on whether I can do what I am about to do on that system.

There are a couple of reasons that I prefer those systems. Probably the most pertinent one is that they have modern hard disk systems attached, and they can read a large file in very short times. The other reason, however, has little to do with disk and file access. It is largely a matter of higher performance in editing a file. The 68XXX systems can search for a string in a very long file, and seemingly go there without waiting. The 68020 in particular is instantaneous in its response when I tell it to do something. Both of the 68XXX systems also have another advantage over the 6809 systems. They have the capabilityofediting VERY large files. I use a buffer of 100K on both those systems, and I can edit the whole PAT editor source file of 66K or so. On the 6809 system, the buffer is just about 29K and there is no hope of increasing that without resorting to virtual memory or paging, something I have never set up on the 6809 systems.

Another consideration for me is that the program that I have written generally gets compiled a number of times in the process of debugging it. I like the 68XXX systems because the compilers compile much faster on them. What takes several minutes on the 6809 takes only one on the 68XXX systems. Here again, if I can do what I have to do on those systems, they are my strong first choice.

You might ask about the IBM clones and how they rate. First (this is primarily my own private strong opinion), let me say that I have not found any editor I consider very good. Some of them splash garbage all over the screen. Others drive me crazy with snow every time I type a character and have annoying features like having to extend my edit file with blank lines before I can add to the end of my text. Still others are large in the extreme though they have enough features so that a user can do almost anything with them. Of course editor evaluation is a HIGHLY SUBJECTIVE activity. What strikes me as extremely useful, you might consider third rate, and vice versa. I like uncluttered screens, use of control keys rather than function keys, and simplicity. Aside from those considerations, the Tandy 1200 exhibits performance substantially poorer than the 2 MHz 6809 system. TSC Extended BASIC is considerably faster than GW BASIC run on the 1200. Running something like MSC's PAL (Finite Element Analysis Program) on the 1200 is agony with waiting for results. The 3000 on the other hand with its 80286 and 80287 co-processor is very fast in the engineering program execution, and had it a good editor and the ability to generate code for my primary work, I would be quite happy with it. As it is, I use it happily to run the software that I need to run that is not available for the 68XXX or 6809 systems, primarily Wintek's SmArtwork for PC board development, MSC's PAL for structural analysis, and, though I don't get involved myself in AutoCad, it is used a great deal of the time for that program, to do PC board component layouts and schematics. The 1200 is downright frustrating for AutoCad, and the 3000 is tolerable.

My preference for the fastest processor, which gives me instant response, brings up an interesting point. Suppose there were two or three other users on that 68020 system? By actual test, just putting another terminalon that system and logging in on it, introduces a fair amount of overhead strain on the system. Put three or four users on that system, and my choice would be to switch to the less expensive 68008 system and be the sole user.

As I said at the beginning of this, I am a bit disappointed at the coming of the end of the first era of microcomputing. The fact is that that era came to an end some time ago for most of the early computer systems, but folks like me who grew up on a system were slow to let go. For example, though Apple has introduced several new lines, the old Apple II continued for a long time to be their bread and butter in spite of their efforts to stop producing the things. In my "work" projects we are rapidly using up the entire memory map of the 6809 and I don't have to tell any computer user that software and its capabilities are getting bigger and not smaller. We will soon outgrow the 64K memory map of the 6809, and when that happens, why bother with any kind of memory paging scheme? There is an advantage in going to a "modern" processor in that we can then advertize "new 16 bit processor for increased performance and capability". We won't go with the 68020 and I won't say that we have the "state of the art" system. What was the state of the art 2 years ago has settled down into being something very useful at a much more reasonable cost.

About here I have to back up and tell a story for many of you who have started subscribing to '68' Micro Journal recently. Those of you who have read this before can skip a few paragraphs. Back when I first got involved in applying 6800 processors to measurement and control applications, we had a single choice of programming languages, assembler. After doing a few programs, admittedly with little style and structure, I decided that my quest would be for a higher level language that would be efficient enough to use for our applications. That is, we couldn't sacrifice a speed factor of ten nor a growth in the size of the object code by a factor of five. The first compilers that I got my hands on were disappointing in both of these respects.

About the time the 6809 came along and we made the switch, things got more promising. The first non-toy Pascal compiler came along, and it was followed by a rush of others. About that time over the period of a year or two, each new compiler that came out seemed to better the performance of the others then available. I watched the execution time drop over that period as programmers learned how to take advantage of the 6809 instruction set, by a factor of just about ten. At the same time, each new compiler generated less and less code to do the functionally identical program. Not only did the new compilers generate less code that ran faster, they did the compiling job faster and faster. It was an exciting time, and I was busily writing columns for this magazine and finding myself with one of everything new that came along, so I could test it. (Thanks, Don).

I learned to program in some "languages" that were absolutely one of a kind, though most of them were subsets of Pascal. As I think back, we had a lot of extended discussion as to "Assembler is best" vs "High Level Language is Best", largely thanks to Dan Farnsworth who admirably defended Assembler. Somewhere in the midst of all this, I learned a good deal about structured programming and writing code modularly, and I realize that these techniques can be applied to Assembler code as well as programs written in Pascal or C, but assembler programmers usually don't make the application (Dan Farnsworth is a notable exception). I argued that the modular programming that made it so easy for Dan to program in assembler was equally applicable to programming in Pascal or C or PL/9 or Whimsical or PASC or Forth, or any number of other languages including BASIC.

When I first switched from assembler to Pascal for my programs, I noted a great reduction in the time it took me to write a program and get it working. Now as I look back, I think it was more the fact that the higher level languages helped me to learn how to program in a structured manner than that it was so much easier to write the program in the higher level language. I say this because now when I do program in assembler, I don't find it nearly as difficult as I did way back before there were any higher level languages. Part of the "HLL is easier" feeling I had was because I was improving my programming skills rapidly about the time I made the switch. Though I think there is less typing involved in writing in a higher level language, some of my readers have disputed that, and I have to admit that for some kinds of programs the difference is not large, though in applications in which there is a lot of number crunching, the higher level language wins by a wide margin.

I bring all this up again because I find myself back in the same situation as several years ago with respect to the 68XXX systems, that is, searching for the best combination of hardware, operating system, and software. First let me say that both of the 68XXX systems run the OS9 operating systems, and that OS9 is a fine operating system for running a computer with multi-user and multi-tasking capabilities. Such a system is ideal for a medium sized company in the area of inventory control or accounting where the primary use is for data storage and retrieval and in cases where a dozen terminals sit in an office and each have about a 5% usage factor. People who need to retrieve data from a computer a few times a day don't mind waiting tens of seconds for an answer on their terminal. At worst, they lose few minutes a day.

On the other hand, two or three workers at the company where I work sit virtually all day and program. The full and undivided attention of a 68XXX processor is not too much power when one of these people is editing or compiling a 50 page program. These very productive and expensive people are paid to write programs, not to wait for computers to finish compiling them. To split the computing capability of the processor even two ways would be foolish. Of course we can purchase more systems and use them in the single user mode with OS9, and that is what we are doing presently. We calculate that if we can increase the throughput of two programmers by 10% by adding another system, the system will be paid for within a year, a pretty good return on investment.

SK*DOS

There is a single user operating system available for the 68008 system. It is SK*DOS by Peter Stark. SK*DOS could be described as a grown up version of FLEX, though it is unrelated to FLEX in terms of authorship. At this point, the problem with the 68008 system and SK*DOS is that there is very little software available to run under SK*DOS. I see the use of that operating system as most practical for development of software for stand-alone applications. With this in mind, I recently wrote a letter to John Spray, who some of you longtime readers might recognize as the author of a very nice language called Whimsical for the 6809. Our company has commissioned John to write us a 68000 version of Whimsical to run under SK*DOS. After we have the bugs out, John will be selling it for use by SK*DOS owners. I am hoping its availability in 6 months or so will help get SK*DOS off the ground.

Incidentally, I don't see OS9 and SK*DOS as competitive products. Each has its place. The capabilities of OS9 are large and it is at the high end of the performance scale when linked with the 68020. Certainly anyone who needs computing power and speed is well advised to go that route. However, there are still a few hobbyists around, who find the 68020 systems and OS9 out of their price range. I don't believe that the hobbyist market has disappeared. I think the depth of that market was greatly overestimated long ago, but there are still numbers of us who work with computers during the day, and would like something of our own with similar capabilities. Put SK*DOS on a 68008 single board computer system such as that made by Peripheral Technology (offered as a complete system, the Mustang 08-A by Data Comp), and you are beginning to reach the price level where hobbyists will again be interested.

Getting a new Operating System off the ground is a difficult thing. Nobody will buy an O.S. for which there is no software, and software writers won't write software for an O.S. that hasn't sold very many copies. It is my hope that this project will start the ball rolling and give SK*DOS a little more momentum. Of course as soon as the compiler is available, I will translate PAT once again so that I can use my favorite editor. There will be at least two other editors available for SK*DOS by then, and there is an Assembler already available. The combination ought to be enough to get things started. It is enough to get those of us who were computer hobbyists back ten years ago, excited all over again.

Editor's Note: *In the near future there will be available a 68000 card that runs SK*DOS and fits inside and on the backplane of an XT/AT clone cabinet. This means it will accept those very inexpensive I/O, hard disk systems and other low priced, mass produced cards and boards that run in those systems. That should make a more economical system available. Also I understand that OS-9 68K will be ported in the near future. I have hope that we can continue to serve those who cannot afford an arm or leg for a 68XXX system. However, if you really want to get down to "rock bottom prices", the Mustang-08A is certainly worth your looking into. With either OS-9 or SK*DOS, or even better - both! And it is here now.*

DMW

EOF

FORTH

A Tutorial Series

By: R. D. Lurie
9 Linda Street

A REAL-WORLD PROBLEM

I thought that I would start off this month with a description of a real-world engineering problem I once had to solve. At the time, I was responsible for establishing computer control of all of the suitable instruments in a testing laboratory devoted to quality control and new product development in a plastics company. The problem at hand was one of taking the data from a liquid chromatography apparatus (it determined the molecular weight distribution of sample of polystyrene). Unfortunately, the environment was very electrically noisy from all of the motors, lights, and other instruments which switched on and off at unpredictable times.

Data were taken with an 8-bit A/D converter in a GIMIX 6800 computer (before the days of the 6809). Since each test sequence ran about 40 minutes, I decided to take a data point every 30 seconds and store each point in an array. The data would be processed and the test results calculated in the 4 minutes between tests. A group of tests was to be started about noon, and run automatically into the night, with the results to be ready the next morning as hard copy.

The technician had been running the tests and tracing the instrument's output on a strip-chart recorder. It took about 70-80 minutes to do each calculation from the data on the recorder, and the reproducibility was barely acceptable. You can see from this the incentive to go with the computer!

Finally, the great day arrived and we were ready to make our first test of the computer system. After the test was run, we looked at the results. Much to our horror and chagrin, the the data taken by the computer looked much like Figure 1.

Of course, everybody, including me, had expected the data to resemble that taken by the strip-chart recorder, and not like shotgun target practice. As soon as I saw the test result, I did what any computer whiz would do—I lied! I assured all and sundry that the problem was a mere detail which would be fixed immediately. Luckily, I was right.

```
list c7-fig
```

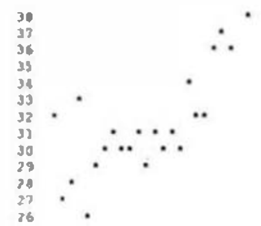
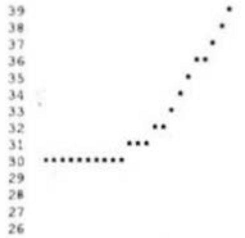
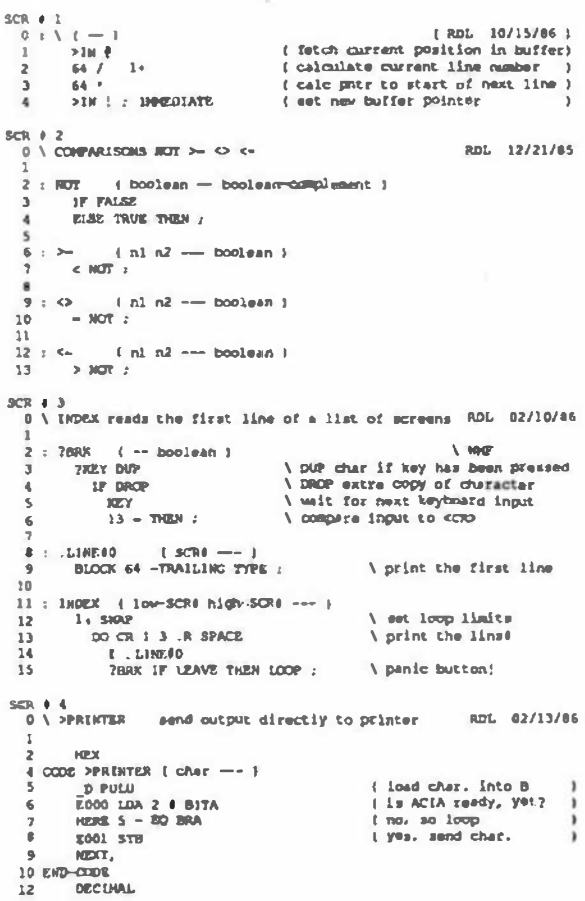


Figure 1--Data as originally received.



After a couple of hours of hard thinking, it dawned on me that the strip-chart recorder had such a slow response time, compared to the computer, that, for practical purposes, it was averaging the data over a finite interval. In this way, it was averaging-out the noise pulses which were masking the data the computer was supposed to be reading. In other words, the computer was, for practical purposes, taking an instantaneous algebraic sum of the true data and what ever noise pulse which happened to occur at the same time. Therefore, to fix the problem, all I had to do was make the computer appear to be a highly dampened strip-chart recorder. I accomplished this by taking 50 measurements, at about 4 per second, and averaging the results. This produced the data shown in Figure 2.

The data in Figure 2 are only a small portion of the whole curve, but I won't bore you with those details, right now. In case you are curious, Figure 2--Data as finally processed.



```
SCR # 1
  0 : \ ( --- )                                 ( RDL  10/15/86 )
  1     >IN @                       ( fetch current position in buffer)
  2     64 /   1+                   ( calculate current line number   )
  3     64 *                        ( calc pntr to start of next line )
  4     >IN ! ; IMMEDIATE           ( set new buffer pointer          )

SCR # 2
  0 \ COMPARISONS NOT >= <> <=                          RDL  12/21/85
  1
  2 : NOT     ( boolean --- boolean-complement )
  3     IF FALSE
  4     ELSE TRUE THEN ;
  5
  6 : >=      ( n1 n2 --- boolean )
  7     < NOT ;
  8
  9 : <>      ( n1 n2 --- boolean )
 10     = NOT ;
 11
 12 : <=      ( n1 n2 --- boolean )
 13     > NOT ;

SCR # 3
  0 \ INDEX reads the first line of a list of screens  RDL  02/10/86
  1
  2 : ?BRK   ( -- boolean )                             \ WMF
  3     ?KEY DUP                    \ DUP char if key has been pressed
  4       IF DROP                   \ DROP extra copy of character
  5         KEY                     \ wait for next keyboard input
  6         13 = THEN ;             \ compare input to <CR>
  7
  8 : .LINE#0      ( SCR# --- )
  9     BLOCK 64 -TRAILING TYPE ;            \ print the first line
 10
 11 : INDEX  ( low-SCR# high-SCR# --- )
 12     1+ SWAP                              \ set loop limits
 13       DO CR I 3 .R SPACE                 \ print the line#
 14         I .LINE#0
 15         ?BRK IF LEAVE THEN LOOP ;        \ panic button!

SCR # 4
  0 \ >PRINTER    send output directly to printer      RDL  02/13/86
  1
  2     HEX
  4 CODE >PRINTER ( char --- )
  5     _D PULU                              ( load char. into B      )
  6     E000 LDA 2 # BITA                    ( is ACIA ready, yet?    )
  7     HERE 5 - EQ BRA                      ( no, so loop            )
  8     E001 STB                             ( yes, send char.        )
  9     NEXT,
 10 END-CODE
 12     DECIMAL
```

the vertical scale is simply 0-255, which is the output from the 8-bit A/D converter. The horizontal scale is just the data count, from 1 to whenever the test was automatically terminated by the computer, based on a complex real-time analysis of the test data. Its only requirement was that it be uniform in time, with no interest at all in the actual amount of time, just its linearity.

What has this got to do with FORTH? Not much, since the program I have been describing was written in BASIC. However, FORTH is often used for this type of application, and I almost did so, but I was still too new to FORTH to have enough confidence in my programming skill to attempt it in FORTH. Still, it is a good example of a software "brute-force" line filter algorithm; and it should be considered as a viable possibility in any instrumentation application.

SOME USEFUL DEFINITIONS

I have had occasion to encounter several undefined words in the FORTH literature. In the majority of cases, I have no idea of the original source, and the definition given is my own version, which may not be the most efficient, but it does work as expected.

The first definition is for \ . This is a somewhat limited replacement for the (...) normally used to enclose a comment. I like it because it is easier to use when you only need one comment on a line. Everything following the \ is treated as a comment by the compiler, so you have to be judicious in its use.

It operates this way. The >IN @ fetches the current character count as of the \ and places this value on the Data Stack. This number is divided by 64 and the integer quotient is left on the Data Stack. This number is incremented by 1 in order to protect against the quotient being 0, as it would be for the first line on the screen. This number is then multiplied by 64 and stored back into >IN . This forces the compiler to conclude that it has already come to the end of a line and to skip down to the beginning of the next line. The 64 is used because that is the normal length of a line on a FORTH screen. Change it to 32 for Sterns' FORTH, for example. The definition must be made IMMEDIATE in order to force execution, instead of compilation.

Only one \ will be effective per line, and there is no terminating delimiter required.

The next 4 definitions should be considered as a group, since they are really extensions of the common comparisons. NOT may not be necessary, as it is often provided, except in the strictest copy of fig-FORTH. However, NOT has often been defined a equivalent to 0= , and that is simply incorrect. If you already have a definition of NOT , then just skip it, and go on to >= , etc.

These three comparisons work by doing the opposite job and then complementing the Boolean result. Their major advantage is in improving the readability of a FORTH definition, rather than doing anything novel.

THE INDEX UTILITY

INDEX is a very useful word missing from FF9 and several other versions of FORTH that I have seen lately. Since it is so useful, I have included here the INDEX that I wrote for my own use. I have modified it slightly to make it easier to understand.

The INDEX utility is used to scan through a list of consecutive screens on disk and print only the first line (line #0). This line is often reserved as a comment line describing what is to be found on the rest of the screen.

INDEX actually consists of two definitions, .LINE#0 and INDEX . The only reason for breaking out .LINE#0 was to make it available to other definitions; otherwise, it could have been left as part of INDEX .

You enter INDEX with the lower and upper screen limits already on the Data Stack, in that order. INDEX first increments the upper screen number by 1, and then exchanges the two numbers on the Data Stack. This places them in the proper order to be recognized by a conventional DO ... LOOP . It also allows us to use the DO ... LOOP index I as the number to be printed as the screen identifier.

Within the DO ... LOOP , the first action is to use CR to send a carriage return/line feed to the display device or printer. The DO ... LOOP index, I , is then called and printed in a 3-column field (this field size will have to be changed if I ever go to a hard-disk).~

Next, I is called again, before transferring operation to .LINE#0 . I has to be called at this time, because the DO ... LOOP index is masked by the return address left by the jump to .LINE#0 . This is a problem often encountered by FORTH beginners, since most documentation does not make it clear that the DO ... LOOP index value is stored on the Return Stack. Therefore, it is not available to subroutines without some non-standard programming gymnastics—a practice to be strongly discouraged!.

.LINE#0 is entered with the screen number on the Data Stack where it is ready for BLOCK . BLOCK reads the indicated screen from the disk (if it is not already in memory), and places the address of the screen buffer on the Data Stack. The 64 is the "count" expected by TYPE . The -TRAILING simply chops off the trailing spaces from the line so that time is not wasted by TYPE in displaying unnecessary characters. Only one line is to be printed, so program control returns to INDEX . The last line has the effect of a "panic button". ?BRK tests the keyboard for any input and returns the appropriate Boolean flag. If the flag was TRUE , the DO ... LOOP execution pauses and waits for another key stroke. A <RETURN> aborts the operation and returns control to the operator; any other key causes operation to resume.

?BRK is not a standard word, so I have included the definition written by Wilson Federici for FF9. ?BRK contains the non- standard word ?KEY, which is defined here in such a way that it can only be used in FF9; however, the standard word ?TERMINAL should produce the same results. Anyone who finds otherwise, please let me know so that I can look for a convenient alternate.

ASSEMBLY LANGUAGE INSIDE FORTH

The last example of FORTH usage involves Assembly language. Most of the time, there is no need for one to use anything but high- level FORTH definitions; however, there are times when it is the lesser of a set of evils.

I encountered such a case when I wanted to send control codes to my printer. FLEX just does not let many control codes through the normal output channel. Therefore, all of my <ESC> sequences were being filtered out before they ever reached the printer; a case of someone protecting me from myself! I don't like that anytime, but, particularly, when it comes to computer programs.

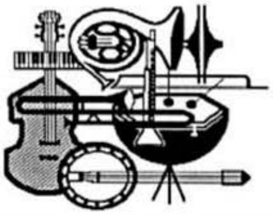
Instead of fighting with FLEX alternate vectors, or some other such machination, I decided to do the job the easy way by bypassing FLEX completely. To do this, I wrote the simple interface software in FORTH Assembly language format shown as >PRINTER . By using this word only for special control codes, I was still able to stay within the spirit of FLEX, but still retain the output flexibility I needed.

This definition of >PRINTER is written in the format required by FF9, but I think that it is pretty much the same for any other version of FORTH for the 68xx family. Notice that the definition opens with the word CODE , instead of the : that we normally expect. This, of course, is to signal the Assembler to process the definition.

Notice, also, that the definition ends with NEXT, END-CODE instead of the familiar ; . The NEXT, is a macro which does essentially the same thing as ; at the end of a definition, and END-CODE is the signal that the Assembler is no longer needed by this definition.

The only other part of the definitton which might not be obvious to everyone is HERE 5 - EQ BRA . This is equivalent to the possibly more familiar mnemonic BEQ *-5 in the infix notation used by Motorola, etc. It means that the branch is to a position 5 bytes before the next program counter value.

BACH ON A BUDGET



An Introduction to Computer Music

by Joseph D. Condon
8072 172nd Street W.
Lakeville, MN 55044

If you would like to experiment with computer generated music but cannot afford to spend alot of money on hardware or software then you should consider implementing this music system. The hardware required for this system can be easily assembled in a couple of hours for less than five dollars and the software is free. This system is capable of reproducing the sound of almost any musical instrument, real or imaginary. You can play a full range of octaves with up to four different notes sounding simultaneously. I have transcribed several Bach and Mozart minuets directly from music store publications in less than a couple of hours with very little effort. These minuets are given in listings 4, 5 and 6. Once the music has been entered into your system using the texteditor, you simply assemble it and then play it.

The software required for the music system is written for the FLEX9 operating system and requires the standard TSC assembler and basic language. The software also requires some type of parallel output port. A printer port will work just fine. The device that actually creates the sound is a simple digital to analog converter that requires no power supply and can be built with a few standard resistors, a capacitor, and a potentiometer. The output from the DAC must then be fed into some type of amplifier and speaker system. The complete schematic forthe DAC is shown in figure 1. You can build the DAC in a seperate box or you can attach it directly to one of your parallel ports inside your computer. How you decide to implement the hardware is entirely up to you. If you are not familiar with your systems hardware, I would recomend that you seek assistance from someone who is.

The sound fidelity of this system is a function of the time required to perform the loop within the "SOUND" subroutine contained in the music macro file. In theory, the highest frequency that can be produced without distortion is equal to one half the reciprocal of the time required to perform the loop within the "SOUND" subroutine. The actual loop time in the "SOUND" subroutine is currently 305 usec for a 1 mhz system clock. This means that the highest frequency that can be produced without distortion is approximately 1639 hz. All frequencies higher than this will contain some amount of distortion. Although the distortion is noticable, it is not totally objectionable.

To generate a music program you will require a minimum of four text files. One of the files describe the musical instruments tonal qualities, another file describes the instruments amplitude envelope. The third file is the music macro file and the fourth file is the actual text of the musical composition.

The tone file is created by a simple basic program given in listing 1. When executed, the program requires you to enter the harmonic composition of the particular instrument that you wish to reproduce. To reproduce the sound of a harpsichord, you might specify five harmonics with amplitudes of 1, .8, .6, .4 and .2 respectively. The program will then create a text file which describes the tonal qualities of the harpsichord.

The envelope file is created by the basic program shown in listing 2. This particular program will generate an envelope file for a harpsichord which has a sharp attack rate followed by a constant linear decay in amplitude. If you want to describe the envelope for a different instrument, you will have to change the program between lines 40 and 60 inclusive.

There have been many books and magazine articles published describing the tone and envelopes generated by different musical instruments therefore I will not attempt to cover the subject in this article. By experimenting and using a little comman sense, you should be able to come up with some very interesting instrument sounds.

The music macro file, shown in listing 3, is the key to the music systems software. It enables the assembler to generate an executable music program from the tone, envelope, and composition text files. The theory and mathametical computations used within and by the macro file become quite complex but it is not necessary for you to understand exactly how it works to be able to use it. The music macro file needs only to be included as a library file within the composition file and the assembler will do all the work for you.

The last file to be described is the actual composition text file. This file tells the assembler what tone and envelope files you wish to use to describe an instruments voice. You may change the instruments voice in the middle of a composition, first part organ, second part piano or whatever. You can change the voice as often as you like during the composition with the "voice" instruction followed by the name of the tone and envelope files you want to use to describe the instruments voice. All tone and envelope files referenced by the voice instruction must be included in the composition file by the use of the assemblers library command.

The first instruction in a composition file should be the "init" instruction. This instruction initializes the DAC port on your system. You will probably have to change the address of the DAC specified in the music macro file to match the address of the PIA port you decide to use on your system. The first instruction in the composition file must also contain a label name as shown in the example.

The "clock" instruction is used to tell the system what speed the processor is running at since this has a direct effect on the pitch of the notes being played. The acceptable values are "1" or "2". If the clock instruction is not specified, the system will default to 1 mhz. The speed of your systems clock will also affect the tempo at which the music will be played. If your systems clock is 2 mhz you should cut the tempo value in half for nonnal play back speed.

The "tempo" instruction describes how quickly the notes in the composition are to be played. The value of the tempo can be between 1 and 9 inclusive. 1 is the slowest and 9 is the fastest. A tempo of 5 is typical.

The "octave" instruction describes the octaves you will be using in the composition. Typically 2,3,4,5 and 6 for a live octave range. Each octave instruction can specify up to 9 octaves and the octave instruction can be used several times if needed, provided you do not specify the same octave more than once. The number of octaves specified have no effect on the size of the program generated but do have a slight effect on the assembly time.

The "note" instruction is used to describe up to four notes in pitch and duration to be played at a specific time. Individual notes are described using the characters a-f followed by a "s" or "f" if the note is a sharp or a flat followed by a value used to indicate the notes octave. The note is then followed by a comma and a value indicating the notes length. When the shortest note of the group completes, the next instruction will be executed. If the next instruction is also a note instruction, the previous notes which have not completed will continue to sound untill they eventually come to there conclusion which may be several note instructions later. This allows the combination of up to four different length notes of different pitch to be sounding at any one time. If a note is changed before it is allowed to complete, it will be chopped off and the new note will begin to sound in its place. In general it is best to think of the four possible notes as four individual sound generators. If you do not want to chop off a note that is still sounding, you should not specify a new value for that sound generator untill its current note has come to completion. You can do this by omitting that note in the next note instruction if the note is the last one to be specified in the note instruction line or you can use the "nul" note description if it is not the last note for the particular note instruction. When playing groups of different length notes, it is best to enter the shortest notes first on the note instruction line.

The length of any note can be 1/64, 1/32, 1/16, 1/8, 1/4, 1/2, 3/4 or 1. The actual amount of time required to play a note is a function of the tempo value specified by the "tempo" instruction. The "rest" instruction simply waits a specified amount of time before continuing on to the next instruction. The rest instruction will also terminate the sounding of the first note specified in the previous note instruction if it has not already come to completion.

The "perform" and "return" instructions are used to repeat a score multiple times without having to reenter the scores text each time you wish to play it. The "stop" instruction is used to terminate the program and return system control to FLEX. The last instruction of any compostion is "end". The end instruction must be followed by the name of the label assigned to the first instruction ("init") described earlier.

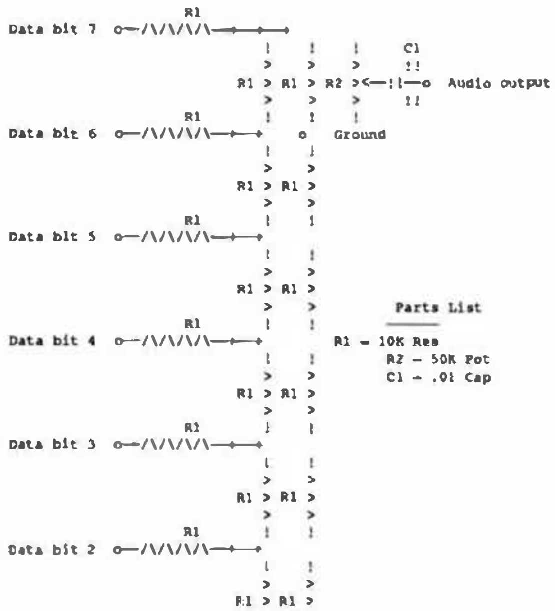
By studying the program examples and experimenting, you should be able to easily master the music programing language. When assembling a music program you should disable the list option using the assembler directive +L in the initiating message. Although the example programs appear to be very short, during assembly time they may expand to over 4 thousand lines of source code. Thats enough to wear out a CRT not to mention a printer.

A word of caution. The main objectives of this music system are maximum flexibility and functionality with minimum hardware and software requirements. This system is by no means intended for someone who is not familiar with their systems hardware or assembly language programing.
+++

LIST 2.FIGURE1

FIGURE #1

Digital to Analog Converter



```
                        R1
Data bit 7  o—/\/\/\/\——+——+——+
                              !    !    !     C1
                              >    >    >     !!
                           R1 > R1 > R2 ><—!!—o  Audio output
                              >    >    >     !!
                        R1    !    !    !
Data bit 6  o—/\/\/\/\——+——+       o   Ground
                              !    !
                              >    >
                           R1 > R1 >
                              >    >
                        R1    !    !
Data bit 5  o—/\/\/\/\——+——+
                              !    !
                              >    >
                           R1 > R1 >
                              >    >              Parts List
                        R1    !    !              ----------
Data bit 4  o—/\/\/\/\——+——+           R1 - 10K Res
                              !    !                R2 - 50K Pot
                              >    >                C1 - .01 Cap
                           R1 > R1 >
                              >    >
                        R1    !    !
Data bit 3  o—/\/\/\/\——+——+
                              !    !
                              >    >
                           R1 > R1 >
                              >    >
                        R1    !    !
Data bit 2  o—/\/\/\/\——+——+
                              !    !
                              >    >
                           R1 > R1 >
```

```
                                    >     >
                        R1          |     |
  Data bit 1   o—/\/\/\/\—+—+       
                                    |     |
                                    >     >
                              R1 >  R1 >
                                    >     >
                        R1          |     |
R1
  Data bit 0   o—/\/\/\/\—+—+—/\/\/\/\—o
Ground

+++
```

LISTING #1

```
10 INPUT "ENTER TONE FILE NAME",N$
20 OPEN NEW N$+".TXT" AS 1
30 DIM W(255)
40 INPUT "ENTER HIGHEST HARMONIC",N
50 DIM A(N),P(N)
60 PRINT "ENTER AMPLITUDE ARRAY"
70 FOR I=1 TO N
80 INPUT A(I)
90 P(I)=2*PI*I/N
100 NEXT I
110 H=0
120 L=0
130 FOR I=0 TO 255
140 W(I)=0
150 FOR J=1 TO N
160 W(I)=W(I)+A(J)*SIN((2*PI/
255)*I*J+P(J))
170 NEXT J
180 IF W(I)=H THEN H=W(I)
190 IF W(I)<L THEN L=W(I)
200 NEXT I
210 M=1/(H-L)
220 FOR I=128 to 255
230 W=W(I)*M
240 PRINT " FCC";INT(63*W)+64
250 PRINT #1, " FCC";INT(63*W)+64
260 NEXT I
270 PRINT N$;
280 PRINT #1,N$;
290 FOR I=0 to 127
300 W=W(I)*M
310 PRINT " FCC";INT(63*W)+64
320 PRINT #1, " FCC";INT(63*W)+64
330 NEXT I
340 CLOSE 1
350 END

+++
```

LISTING #2

```
10 INPUT "ENTER ENVELOPE FILE
NAME",N$
20 OPEN NEW N$+".TXT" AS 1
30 DIM W(255)
40 FOR I=0 to 255
50 W(I)=(255-I)/2
60 NEXT I
70 FOR I=128 TO 255
80 PRINT " FCC";INT(W(I))
90 PRINT #1," FCC";INT(W(I))
100 NEXT I
110 PRINT N$;
120 PRINT #1,N$;
130 FOR I=0 to 127
140 PRINT " FCC";INT(W(I))
150 PRINT #1," FCC";INT(W(I))
160 NEXT I
170 CLOSE 1
180 END

+++
```

LISTING #3

```
 opt exp
*(=================================)*
*(
)*
*(              Music Programing
Language
)*
*(
)*
*(                        by
)*
*(                     Joe Condon
)*
*(
)*
*(                      01/29/85
)*
*(
)*
*(=================================)*

DAC equ $E010
WARMS equ $CD03

CLK set 1
TMPO set 5
PREID set 256

 org 0

*(=================================)*
*(  Temporary Storage for Play
Subroutine
)*
*(=================================)*

wpnt rmb 2
epnt rmb 2

exitf fcb 0
sound fcb 0

wincl fdb 0
widxl fdb 0
eincl fdb 0
eidxl fdb 0
maskl fcb 0

winc2 fdb 0
widx2 fdb 0
einc2 fdb 0
eidx2 fdb 0
mask2 fcb 0

winc3 fdb 0
widx3 fdb 0
einc3 fdb 0
eidx3 fdb 0
mask3 fcb 0

winc4 fdb 0
widx4 fdb 0
einc4 fdb 0
eidx4 fdb 0
mask4 fcb 0

*(=======================*)
*(  Play Note Subroutine  *)
*(       (305 us loop)    *)
*(=======================*)

play clr exitf
     ldx wpnt
     ldy epnt

play1 lda widxl
      lda a,x
      ldb eidxl
      ldb b,y
      mul
      anda maskl
      sta sound

      lda widx2
      lda a,x
      ldb eidx2
      ldb b,y
      mul
      anda mask2
      adda sound
      sta sound

      lda widx3
      lda a,x
      ldb eidx3
      ldb b,y
      mul
      anda mask3
      adda sound
      sta sound

      lda widx4
      lda a,x
      ldb eidx4
      ldb b,y
      mul
      anda mask4
      adda sound
      sta DAC

      ldd widxl
      addd wincl
      std widxl
      ldd eidxl
      addd eincl
      std eidxl
      bcc play2
      clr maskl
      clr eincl
      clr eincl+1
      inc exitf

play2 ldd widx2
      addd winc2
      std widx2
      ldd eidx2
      addd einc2
      std eidx2
      bcc play3
      clr mask2
      clr einc2
      clr einc2+1
      inc exitf

play3 ldd widx3
      addd winc3
      std widx3
      ldd eidx3
      addd einc3
      std eidx3
      bcc play4
      clr mask3
      clr einc3
      clr einc3+1
      inc exitf

play4 ldd widx4
      addd winc4
      std widx4
      ldd eidx4
      addd einc4
      std eidx4
      bcc play5
      clr mask4
      clr einc4
      clr einc4+1
      inc exitf

play5 tst exitf
      lbeq play1
      rts

*(========================)*
*(  Initialize DAC Macro   )*
*(========================)*

init macro
     clr DAC+1
     lda #$FF
     sta DAC
     lda #$04
     sta DAC+1
     endm

*(================)*
*(  Clock Macro    )*
*(================)*

clock macro
CLK   set &1
      endm

*(================)*
*(  Scale Macro    )*
*(================)*

scale  macro
SMUL   set 1
       dup &1-1
SMUL   set SMUL*2
       endd
SMUL   set SMUL/CLK
c&1    set SMUL*654
cs&1   set (c&1+d&1)/2
df&1   set (c&1+d&1)/2
d&1    set SMUL*734
ds&1   set (d&1+e&1)/2
ef&1   set (d&1+e&1)/2
e&1    set SMUL*824
f&1    set SMUL*873
fs&1   set (f&1+g&1)/2
gf&1   set (f&1+g&1)/2
g&1    set SMUL*979
gs&1   set (g&1+a&1)/2
af&1   set (g&1+a&1)/2
a&1    set SMUL*1099
as&1   set (a&1+b&1)/2
bf&1   set (a&1+b&1)/2
b&1    set SMUL*1234
       endm

*(=================)*
*(  Octave Macro    )*
*(=================)*

octave macro
       ifnc &1,
```

```
         scale &1
         ifnc &2,
           scale &2
           ifnc &3,
             scale &3
             ifnc &4,
               scale &4
               ifnc &5,
                 scale &5
                 ifnc &6,
                   scale &6
                   ifnc &7,
                     scale &7
                     ifnc &8,
                       scale &8
                       ifnc &9,
                         scale &9
                       endif
                     endif
                   endif
                 endif
               endif
             endif
           endif
         endif
       endif
       endm

*(================)*
*(  Tempo Macro    )*
*(================)*

tempo macro
      ifc &1,1
TMPO    set 1
PRELD   set 64
      endif
      ifc &1,2
TMPO    set 2
PRELD   set 256
      endif
      ifc &1,3
TMPO    set 3
PRELD   set 448
      endif
      ifc &1,4
TMPO    set 4
PRELD   set 256
      endif
      ifc &1,5
TMPO    set 5
PRELD   set 256
      endif
      ifc &1,6
TMPO    set 6
PRELD   set 1024
      endif
      ifc &1,7
TMPO    set 7
PRELD   set 1024
      endif
      ifc &1,8
TMPO    set 8
PRELD   set 1024
      endif
      ifc &1,9
TMPO    set 9
PRELD   set 1600
      endif
      endm

*(================)*
*(  Voice Macro    )*
*(================)*

voice macro
      ldd #&1
      addd #128
      std wpnt
      ldd #&2
      addd #128
      std epnt
      endm

*(================)*
*(  Notel Macro    )*
*(================)*

notel macro
      ldd #&2
      std winc&1
      ldd #TMPO*192/(64*&3)
      std einc&1
      ldd #PRELD
      std widx&1
      std eidx&1
      lda #$FF
      sta mask&1
      endm

*(================)*
*(  Note Macro     )*
*(================)*

note' macro
     ifnc &1,
       ifnc &1,nul
         notel 1,&1,&2
       endif
       ifnc &3,
         ifnc &3,nul
           notel 2,&3,&4
         endif
         ifnc &5,
           ifnc &5,nul
             notel 3,&5,&6
           endif
           ifnc &7,
             ifnc &7,nul
               notel 4,&7,&8
             endif
           endif
         endif
       endif
       lbsr play
     endif
     endm

*(================)*
*(  Rest Macro     )*
*(================)*

rest macro
     ldd #TMPO*192/(64*&1)
     std eincl
     ldd #PRELD
     std eidxl
     clr maskl
     lbsr play
     endm

*(=================)*
*(  Perform Macro   )*
*(=================)*

perform macro
```

```
        lbar &1
        endm

*(====================)*
*(  Return Macro      )*
*(====================)*

return macro
       rta
       endm

*(=========================)*
*(  Stop Program Macro     )*
*(=========================)*

stop macro
     jmp WARMS
     endm

*(=========================)*

+++
```

LISTING #4

```
*(==================================)*
*(
)*
*(               Minuet In  G
)*
*(
)*
*(                   by
)*
*(
)*
*(            Johann Sebastian Bach
)*
*(
)*
*(==================================)*

 lib music   ***  music macro file
name
 lib ht      ***  harpsichord tone
file name
 lib he      ***  harpsichord
envelope file name

*(==================================)*
*(            Start of music
program
)*
*(==================================)*

minuet init
       clock 1
       voice ht,he
       octave 2,3,4,5
       tempo 5
       perform part1
       perform part1
       perform part2
       perform part2
       atop

*(==================================)*

part1  note d5,1/4,e3,1/2,g3,1/
2,b3,1/2
       note g4,1/8
```

```
note a4,1/8
note b4,1/8,f3,1/4
note c5,1/8

note d5,1/8,g3,3/4
rest 1/8
note g4,1/8
rest 1/8
note g4,1/8
rest 1/8

note e5,1/4,a3,3/4
note c5,1/8
note d5,1/8
note e5,1/8
note fs5,1/8

note g5,1/8,g3,3/4
rest 1/8
note g4,1/8
rest 1/8
note g4,1/8
rest 1/8

note c5,1/4,f3,3/4
note d5,1/8
note c5,1/8
note b4,1/8
note a4,1/8

note b4,1/4,e3,3/4
note c5,1/8
note b4,1/8
note a4,1/8
note g4,1/8

note f4,1/4,b3,1/4
note g4,1/8,g3,1/4
note a4,1/8
note b4,1/8,e3,1/4
note g4,1/8

note b4,1/4,b3,1/4
note a4,1/2,b2,1/8
note nul,0,a3,1/8
note nul,0,g3,1/8
note nul,0,f3,1/8

note d5,1/4,g3,1/2
note g4,1/8
note a4,1/8
note b4,1/8,f3,1/4
note c5,1/8

note d5,1/8,e3,1/8
rest 1/8
note g4,1/8,g3,1/8
rest 1/8
note g4,1/8,e3,1/8
rest 1/8

note e5,1/4,a3,3/4
note c5,1/8
note d5,1/8
note e5,1/8
note fa5,1/8

note g5,1/8,g3,1/4
rest 1/8
note g4,1/8,a3,1/8
note nul,0,g3,1/8
note g4,1/8,f3,1/8
```

```
       note nul,0,e3,1/8

       note c5,1/4,f3,1/2
       note d5,1/8
       note c5,1/8
       note b4,1/8,ds3,1/4
       note a4,1/8

       note b4,1/4,e3,1/2
       note c5,1/8
       note b4,1/8
       note a4,1/8,g3,1/4
       note g4,1/8

       note a4,1/4,a3,1/4
       note b4,1/8,b3,1/4
       note a4,1/8
       note g4,1/8,b2,1/4
       note f4,1/8

       note g4,3/4,e3,1/2
       note nul,0,e2,1/4

       return

*(==================================)*

part2  note b5,1/4,e3,3/4
       note g5,1/8
       note a5,1/8
       note b5,1/8
       note g5,1/8

       note a5,1/4,ds3,3/4
       note d5,1/8
       note e5,1/8
       note fs5,1/8
       note d5,1/8

       note g5,1/4,c3,1/4
       note e5,1/8,e3,1/4
       note fs5,1/8
       note g5,1/8,c3,1/4
       note d5,1/8

       note cs5,1/4,f3,1/2
       note b4,1/8
       note cs5,1/8
       note a4,1/4,f2,1/4

       note a4,1/8,f3,3/4
       note b4,1/8
       note cs5,1/8
       note d5,1/8
       note e5,1/8
       note fs5,1/8

       note g5,1/8,g3,1/8
       rest 1/8
       note fa5,1/8,b3,1/8
       rest 1/8
       note e5,1/8,as3,1/8
       rest 1/8

       note fs5,1/8,b3,1/8
       rest 1/8
       note a4,1/8,ds3,1/8
       rest 1/8
       note cs5,1/8,f3,1/8
       rest 1/8

       note d5,3/4,b3,1/4
       note nul,0,b2,1/4
```

```
        note nul,0,a3,1/4

        note d5,1/4,g3,1/4
        note g4,1/8,b3,1/4
        note f4,1/8
        note g4,1/4,g3,1/4

        note e5,1/4,a3,1/4
        note g4,1/8,c4,1/4
        note f4,1/8
        note g4,1/4,a3,1/4

        note d5,1/4,g3,1/4
        note c5,1/4,f3,1/4
        note b4,1/4,e3,1/4

        note a4,1/8,b3,1/2
        note g4,1/8
        note f4,1/8
        note g4,1/8
        note a4,1/4

        note d4,1/8,b2,3/4
        note e4,1/8
        note f4,1/8
        note g4,1/8
        note a4,1/8,ds3,1/4
        note b4,1/8

        note c5,1/8,c3,1/8
        rest 1/8
        note b4,1/8,e3,1/8
        rest 1/8
        note a4,1/8,ds3,1/8
        rest 1/8

        note b4,1/8,e3,1/4
        note d5,1/8
        note g4,1/8,g2,1/8
        rest 1/8
        note f4,1/8,b2,1/8
        rest 1/8

        note e3,1/8,b4,3/4,d4,3/
4,g4,1/2
        rest 1/8
        note b2,1/8
        rest 1/8
        note e2,1/8
        rest 1/8

        return

*(--------------------------------)*

        end minuet

+++
```

LISTING #5

```
*(--------------------------------)*
*(
)*
*(              Minuet In F
)*
*(
)*
*(                  by
)*
*(
)*
*(        Wolfgang Amedeus Mozart
)*
*(
)*
*(--------------------------------)*

 lib music    ***  music macro file name
 lib ht       ***  harpsichord tone file
name
 lib he       ***  harpsichord envelope
file name

*(--------------------------------)*
*(            Start of music program
)*
*(--------------------------------)*

minuet init
       voice ht,he
       octave 2,3,4,5
       tempo 5

       perform part1
       perform part1

       note c5,1/8,ds3,3/4
       note ef5,1/8
       note a4,1/8
       rest 1/8
       note a4,1/8
       rest 1/8

       note bf4,1/8,e3,3/4
       note d5,1/8
       note g4,1/8
       rest 1/8
       note g4,1/8
       rest 1/8

       note a4,1/8,a3,1/4
       note c5,1/8
       note fs4,1/8,b3,1/8
       rest 1/8
       note fs4,1/8,b2,1/8
       rest 1/8

       note fs4,1/2,e3,1/4
       note nul,0,b2,1/4
       note g4,1/4,e2,1/4

       note bf4,1/8,c3,3/4
       note d5,1/8
       note g4,1/8
       rest 1/8
       note g4,1/8
       rest 1/8

       note a4,1/8,d3,3/4
       note c5,1/8
       note f4,1/8
       rest 1/8
       note f4,1/8
       rest 1/8

       note g4,1/8,g3,1/4
       note bf4,1/8
       note e4,1/8,a3,1/8
       rest 1/8
       note e4,1/8,a2,1/8
       rest 1/8

       note e4,1/2,d3,1/4
       note nul,0,a2,1/4
       note f4,1/4,d2,1/4

       note f5,1/8,f3,3/4
       note a5,1/8
       note c5,1/8
       rest 1/8
       note c5,1/8
       rest 1/8

       note d5,1/8,g3,3/4
       note f5,1/8
       note bf4,1/8
       rest 1/8
       note bf4,1/8
       rest 1/8

       note a4,1/8,a3,1/2
       note c5,1/8
       note f4,1/8
       rest 1/8
       note f4,1/8,a2,1/4
       rest 1/8

       note e4,1/2,b2,3/4
       note f4,1/4

       note f5,1/8,f3,3/4
       note a5,1/8
       note c5,1/8
       rest 1/8
       note c5,1/8
       rest 1/8

       note d5,1/8,g3,3/4
       note f5,1/8
       note bf4,1/8
       rest 1/8
       note bf4,1/8
       rest 1/8

       note a4,1/8,a3,1/2
       note c5,1/8
       note f4,1/8
       rest 1/8
       note e4,1/8,a2,1/4
       rest 1/8

       note e4,1/2,d3,1/4
       note nul,0,a2,1/4
       note f4,1/4,d2,1/4

       stop

*(--------------------------------)*

part1  note f5,1/8,d3,1/2
       note a5,1/8
       note c5,1/8
       rest 1/8
       note c5,1/8,f3,1/4
       rest 1/8

       note d5,1/8,g3,1/2
       note f5,1/8
       note bf4,1/8
       rest 1/8
       note bf4,1/8,g3,1/4
       rest 1/8

       note a4,1/8,a3,1/2
       note c5,1/8
       note f4,1/8
       rest 1/8
       note e4,1/8,a2,1/4
       rest 1/8

       note e4,1/2,d3,1/4
       note nul,0,a2,1/4
       note f4,1/4,d2,1/4

       note c4,1/8,a2,3/4
       note e4,1/8
       note g4,1/8
       rest 1/8
       note g4,1/8
       rest 1/8

       note c4,1/8,a2,3/4
       note f4,1/8
       note a4,1/8
       rest 1/8
       note a4,1/8
       rest 1/8
```

```
        note C4,1/8,a2,1/4
        note g4,1/8
        note bf4,1/8,c3,1/8
        rest 1/8
        note a4,1/8,d3,1/8
        rest 1/8

        note a4,1/2,a3,1/4
        note nul,0,a3,1/4
        note g4,1/4,a2,1/4

        return

*{-----------------------------------}*

        end minuet

+++
```

LISTING #6

```
*{-----------------------------------}*
*{
}*
*{               Minuet In D Minor
}*
*{
}*
*{                      by
}*
*{
}*
*{            Johann Sebastian Bach
}*
*{
}*
*{-----------------------------------}*

 lib music   ***  music macro file name
 lib ht      ***  harpsichord tone file
name
 lib he      ***  harpsichord envelope
file name

*{-----------------------------------}*
*{            Start of music program
}*
*{-----------------------------------}*

minuet init
        clock 1
        voice ht,he
        octave 1,2,3,4,5
        tempo 5

        note a4,1/4,a2,1/2
        note f5,1/8
        note e5,1/8
        note d5,1/8,c3,1/4
        note cs5,1/8

        note d5,1/4,d3,1/2
        note a4,1/4
        note bf4,1/4,e3,1/4

        note cs4,1/8,f3,1/2
        note e4,1/8
        note g4,1/8
        note bf4,1/8
        note a4,1/8,f2,1/4
        note g4,1/8

        note f4,1/4,a2,1/4
        note e4,1/8,f2,1/4
        note f4,1/8
        note d4,1/4,b2,1/4

        note f4,1/4,b2,1/2
        note bf4,1/8
        note a4,1/8
        note d5,1/8,gf2,1/4
        note c5,1/8

        note f5,1/4,f2,1/2
        note e5,1/8
        note d5,1/8
        note c5,1/8,c3,1/4
        note bf4,1/8

        note a4,1/8,d3,1/4
        note bf4,1/16
        note c5,1/16
        note f4,1/8,f2,1/8
        rest 1/8
        note e4,1/8,a2,1/8
        rest 1/8

        note d2,1/4,f4,3/4
        note d3,1/8
        note c3,1/8
        note b2,1/8
        note as2,1/8

        note a4,1/4,b2,1/2
        note f5,1/8
        note e5,1/8
        note d5,1/8,c3,1/4
        note cs5,1/8

        note d5,1/4,d3,1/2
        note a4,1/4
        note bf4,1/4,e3,1/4

        note cs4,1/8,f3,1/2
        note e4,1/8
        note g4,1/8
        note bf4,1/8
        note a4,1/8,f2,1/4
        note g4,1/8

        note f4,1/4,b2,1/4
        note e4,1/8,f2,1/4
        note f4,1/8
        note d4,1/4,b2,1/4

        note f4,1/4,b2,1/4
        note bf4,1/8,a2,1/4
        note a4,1/8
        note d5,1/8,gf2,1/4
        note c5,1/8

        note f5,1/4,f2,1/2
        note e5,1/8
        note d5,1/8
        note c5,1/8,c3,1/4
        note bf4,1/8

        note a4,1/8,d3,1/4
        note bf4,1/16
        note c5,1/16
        note f4,1/8,f2,1/8
        rest 1/8
        note e4,1/8,a2,1/8
        rest 1/8

        note d3,1/4,f4,3/4
        note a2,1/4
        note d2,1/4

        perform part2
        perform part2

        stop

*{-----------------------------------}*

part2   perform part3
        note b2,1/4,d5,3/4
        note b1,1/8
        note f2,1/8
        note b2,1/8
        note c3,1/8

        perform part3

        note b2,1/4,d5,3/4
        note f2,1/4
        note b1,1/4

        return

*{-----------------------------------}*

part3   note a5,1/8,d3,1/8
        rest 1/8
        note f4,1/8,f3,1/4
        note a5,1/8
        note g5,1/8,g3,1/4
        note f5,1/8

        note e5,1/16,a3,3/8
        note f5,1/16
        note g5,1/8
        note c5,1/2
        note nul,0,e3,1/8
        note nul,0,c3,1/8
        note nul,0,a2,1/8

        note f5,1/8,b2,1/8
        rest 1/8
        note d4,1/8,a2,1/4
        note f5,1/8
        note e5,1/8,g2,1/4
        note d5,1/8

        note cs5,1/16,f2,1/8
        note d5,1/16
        note e5,1/8,g2,1/8
        note as2,1/8,a4,1/2
        note b2,1/8
        note c3,1/8
        note d3,1/8

        note a4,1/8,e3,1/4
        note b4,1/8
        note cs5,1/8,c3,1/4
        note d5,1/8
        note e5,1/8,as2,1/4
        note f5,1/8

        note g5,1/8,f2,1/4
        note e5,1/8
        note cs5,1/8,f3,1/4
        note bf5,1/8
        note a5,1/8,as3,1/4
        note g5,1/8

        note f5,1/16,b3,1/4
        note e5,1/16
        note d5,1/8
        note e5,1/8,e3,1/8
        rest 1/8
        note cs5,1/8,f3,1/8
        rest 1/8

        return

*{-----------------------------------}*

        end minuet

+++
```

Bit-Bucket

By: All of us

"Contribute Nothing - Expect Nothing", DMW '86

MOTOROLA INC.

Microprocessor Products Group
6501 William Cannon Drive West
Austin, Texas 78735.6598

EDITORIAL CONTACT:
Alan Kelly
Cunningham Communication, Inc.
2350 Mission College Boulevard
Suite 900
Santa Clara, CA 95054
(408) 982-0400

READER CONTACT:
Dean Mosley
(512) 440-2839

INQUIRY RESPONSE:
Technical Information Center
P.O. Box 52073
Phoenix, AZ 85072

MOTOROLA ANNOUNCES DEVELOPMENT OF 25 MHz 68030

Sampling to Begin December 1987

NEW YORK, Oct. 29, 1987—In tandem with the shipping of its newest 32-bit microprocessor, the 68030 (030) at 20 MHz speed, Motorola today announced the development of a faster 25 MHz version, scheduled for sampling in December 1987.

The 030 (nicknamed "oh thirty") can provide twice the performance of Motorola's 68020, the most widely used 32-bit chip in the world. Its advanced features include on-chip instruction and data caches, on-chip memory management and parallel architecture typical of mainframe computers.

The speed of a microprocessor is a measure not only of how quickly a chip can perform an operation; it is also a measure of the efficiency by which a chip is manufactured. For example, Motorola's 68020 was initially manufactured at speeds of 12.5 and 16 MHz, but as the manufacturing process matured the 020 was able to reach clock speeds of 20 and 25 MHz.

"The longer a chip line is manufactured, the faster it gets," said Dr. Murray A. Goldman, senior vice president and general manager of the Microprocessor Products Group, (Austin, Texas). "All 68000 chips undergo a similar manufacturing process because the 68000 family is based on a completely compatible architecture. Whether we're making 68000s or 68030s, we use the same fabrication process. With the maturity of the eight-year-old 68000 line we'll produce 030s at speeds our competition can't touch. The Intel 80386, for example, is just now beginning to ship at 20 MHz. What's more, the 386 is a first-generation product that is comparable to Motorola's previous-generation 68020. With the introduction of the 030 at 25 MHz, Motorola is two big steps ahead."

According to Motorola, the 030 will be available at speeds above 25 MHz later in 1988.

Motorola's $2 billion Semiconductor Products Sector (Phoenix, Ariz.), which includes the Microprocessor Products Group (Austin, Texas), is a division of Motorola Inc. It is the largest and broadest semiconductor supplier in North America with a balanced product portfolio of over 50,000 devices.

MOTOROLA ANNOUNCES 68030 EMULATOR MODULE

Emulator Will Speed 030-Based Products to Market

NEW YORK, Oct. 29, 1987—Motorola today announced the availability of an emulator for the 68030 (030) microprocessor. The new 030 emulator module provides system design support that will speed 030-based hardware and software products to market.

The 030 (nicknamed "oh thirty"), also unveiled today in New York, is the newest member of the 32-bit 68000 family and is fully compatible with its predecessors, the 68000, 68010 and 68020. Providing the 030 emulator at the same time that the chip is delivered makes easy development of 030-based products possible for system designers of all sizes.

Microprocessor emulation is key to successful product development. Emulators allow product designers to test their hardware and software without building expensive prototypes as well as to allow multiple engineers to work on development projects. All of these factors are crucial to speeding products to market.

The emulator module offers the same functionality as the 030 while giving computer design engineers important information on how to optimize 030-based system performance and design. For instance, the module can run time trials or benchmarks of a specified target application in multiple configurations. This eliminates the need to build and test prototype systems to make design tradeoff decisions. After determining the best design approach for a specific application, designers can use the 030 emulator for additional hardware and software development and testing.

"The 030 emulator will help bring hardware and software products to market quickly, and we are happy to announce it at the same time the 68030 becomes available," said Dr. Murray A. Goldman, senior vice president and general manager of Motorola's Microprocessor Products Group (Austin, Texas). "We provide the world with not only the highest performance chips, but also with tools that allow their power to be maximized."

The 030 emulator is part of the HDS-300 Hardware/Software Development Station from Motorola that is offered for all microprocessors in the 68000 family. The HDS-300 is a host-independent interface that connects most computers to the 030 emulator module. This allows multiple engineers to use the emulator concurrently, further accelerating products to market.

Key Features

Technical features of the new emulator include the following:

- Zero wait-state emulation for target memory at 25 MHz.
- Zero wait-state emulation for internal emulation memory at 20 MHz and a maximum of one wait-state at 25 Mhz.
- Synchronous signal generation that may be used to optimize performance of target synchronous bus timing.
- Three hardware breakpoints that can halt emulation before execution of any instruction located anywhere in memory, including instructions located in ROM.
- Sixty-four software breakpoints for program execution control and debugging.
- Support of 030 synchronous and burst access memory.
- Emulation memory of 64K bytes, with options for 256K bytes and 1M byte.
- Trace capability is provided by the optional System Performance Analyzer, which captures qualified events in a trace buffer that is 160 bits wide and 4096 events deep.
- Emulation memory that can be mapped on any 4K byte address in blocks as small as 4K bytes. Memory can also be mapped in increments of 8–, 16– or 32–bit wide asynchronous memory with 0 to 7 wait states, or 32-bit wide synchronous memory.

Orders are now being accepted for delivery of the 030 emulator module. Price ranges from $9,400 to $14,900, depending on the amount of emulation memory purchased with the module. The HDS-300 is available for $7,200.

Motorola's $2 billion Semiconductor Products Sector (Phoenix, Ariz.), which includes the Microprocessor Products Group (Austin, Texas), is a division of Motorola Inc. It is the largest and broadest supplier of semiconductors in North America, with a balanced product portfolio of over 50,000 devices.

Tandy Corporation/Radio Shack

News Release

For Release: IMMEDIATE RELEASE

For Further Information Contact:
Fran McGehee
Marketing Information
817-390-3487

RADIO SHACK RELEASES OS-9 DEVELOPMENT SYSTEM SOFTWARE FOR COLOR COMPUTER 3

Radio Shack®, a division of Tandy Corporation and major retailer of computer and electronic products, has released the OS-9™ Development System for the Color Computer 3™.

The OS-9 Development System is a complete editor/assembler with full-screen editing and specialty I/O drivers. The development system gives serious as well as novice Color Computer programmers tools to develop software programs for their own customized applications.

The OS-9 Level Two operating system (Cat. No. 26-3031) is required to take advantage of programming tools provided by the development system.

Available through more than 7,000 Radio Shack Stores, Radio Shack Computer Centers® and participating dealers nationwide, the suggested retail price of the OS-9 Development System (Cat. No. 26-3032) is $99.95.

- 30 -

Radio Shack and Radio Shack Computer Centers are registered service marks of Tandy Corporation.
Color Computer 3 is a trademark of Tandy Corporation.
OS-9 is a trademark of Microware Systems Corporation.

WINDRUSH
Micro Systems Ltd

[illegible] Laboratories Reg. Office
North Walsham, Norfolk NR28 9SA
Tel (0692) 404086
Telex 975548 WMICRO G
MAPCON approved consultants

WINDRUSH HARDWARE & SOFTWARE RESCUE SERVICE

Windrush Micro Systems Limited announces the immediate availability of access to a team of Motorola MC6809, MC68000 and MC68020 design specialists.

Design engineers are now facing formidible learning curves in the design and programming of systems using the latest generation of 16-bit and 32-bit microprocessors. This often results in projects taking two to three times the original project estimates.

Windrush offer a full range of services ranging from project start-up assistance to complete design, prototyping, programming and manufacture.

Windrush have specialised in the Motorola and Hitachi 68XXx family since the introduction of the MC6800 in 1975. The company has not diluted its expertise by involvement with the Intel, Rockwell or Zilog (et al) families of processors. Our expertise ranges from the application of single chip micros, eg 6801, 6803, 6805 and 6811 to the latest generation of 32 bit processors, eg 68020 and 68030.

The company also specialises in the application of microprocessors in microprocsssor development systems using FLEX and OS-9/68k, process/machine tool control systems and data logging equipment.

We ***always*** offer fixed cost quotes. There are ***never*** any hidden costs. You know, *in* ***advance***, exactly how much the project is going to cost, right down to the last penny.

Before you start an in-house project why not give us a call ... you may find that we can do the job more economically and in a more timely manner. Important factors if you want to beat your competition.

For further information contact Bill Dickinson on (0692) 404086

GESPAC Inc.

50 West Hoover Ave.
Mesa, Arizona 85202
Tel. (602) 962-5559
Fax. (602) 962-5750

Reader Contact: Mark Stephens
Editorial Contact: Cosma Pabouctsidis

FOR IMMEDIATE RELEASE

GESPAC INTRODUCES 2 MEGABYTE DRAM BOARD FOR G-64 BUS

MESA, AZ, NOVEMBER 25, 1987--GESPAC introduces a 2 megabyte Dynamic RAM board built on a single height Eurocard and compatible with the standard G-64 bus.

The GESRAM-14A is organized as 1 Meg of 16-bit words, with on-board parity checking to protect against system failures. The GESRAM-14A uses asynchronous timing and provides an access time of 240 ns maximum.

The board features a very high level of integration, using ZIP packages and surface mounted IC's. The GESRAM-14A will work with all 68000, 68020 and 80286 processor boards produced by GESPAC.

The board is available today and is priced at a low unit price of $995.

Industrial Keyboard Display Interface for the G-64 bus

Mesa, AZ, November 6, 1987--GESPAC has introduced a versatile keyboard/display interface board for the G-64 bus. The GESIKD-2 is specifically designed to attach to a variety of compact keyboard/display assemblies to provide cost effective human interface in machine control and instrumentation applications.

The GESIKD-2 allows the system integrator to connect a standard or custom keypad/keyboard of up to 90 keys plus the shift and control keys. The board's output is designed to drive today's most popular text or graphics Liquid Crystal Display (LCD) or Vacuum Fluorescent Display (VFD). Among the display units supported are those manufactured by Hitachi, Philips, Epson, and Futaba.

The GESIKD-2 has an on-board DC/DC converter with an adjustable output of -5 to -22 Volts that provides the necessary negative voltage (VEE) for LCD displays. The board also comes with an optional DC to AC converter to drive electro luminescent (EL) lamp for backlit LCD displays.

The GESIKD-2 is designed on a standard single height Eurocard and can operate with any 8, 16 or 32-bit microprocessor on the G-64 bus. The G-64 bus is a second generation 16-bit bus aimed at midrange industrial applications.

The GESIKD-2 is available today at the basic unit price of $395.

First 68030 Single Board Computer Offers 16.7 and 30MHz Zero Wait State Operation For Performance Sensitive Real-Time Applications

Most Powerful 32-bit Microprocessor Drives Multiprocessing VME/PLUS™ Board

Force Computers GmbH,
Daimlerstrasse 9, D-8012
Ottobrunn, West Germany

Contact:
Force USA: Wayne Fischer (408) 354-3410
Force GmbH: Anton Nausch (089) 600-910

LOS GATOS, CA., November 17, 1987 — The first working 68030 single board computer for the VMEbus, and perhaps the first off-the-shelf 68030-based product to reach the worldwide marketplace was announced today by Force Computers.

"We demonstrated a working production model of the new CPU-32 board five weeks ago at *Systems '87* in Munich," said Martin Weisberg, Force Executive Vice President. The CPU-32 provides 32-bit VMEbus and VME Subsystem Bus (VSB) interfaces while achieving zero wait state performance in a single card cage slot. Using an advanced gate array and dense surface mount technology, the CPU-32 combines the computational logic of three boards into one. The 68030 imparts processing power that takes the CPU-32 beyond any known single board computer. The microprocessor itself includes an on-chip paged memory management unit (PMMU) capable of demand-paged control of up to 4 gigabytes of virtual memory. The board includes a 68882 floating point coprocessor.

The CPU-32 design is based on the recently announced VME/PLUS architecture for high-end real-time applications. Supporting the 68030 is 1Mbyte of 100ns static RAM. Versions rated for operation at 16.7 and 20MHz are being prepared for shipment during the first quarter of 1988, pending microprocessor availability. "If our microprocessor vendor delivers, we will deliver," said Wayne Fischer, Director of Marketing.

All versions of the CPU-32 are equipped with VMEPROM, the recently announced PDOS™ real-time operating system kernel plus FORCEbug™ monitor/debugger software, at no additional charge..

The CPU-32 is one of the first VMEbus products to ship with a gate array solution to VMEbus interface and control. A 132-pin CMOS gate array replaces dozens of integrated circuits. "As a VME/PLUS product, the CPU-32 is intended for real-time applications where the memory management capabilities of the 68030 processor are needed in combination with uncompromising performance," said Weisberg. "The gate array solution helps reduce board count from 3 to 1, while the remarkable 68030 microprocessor gives our customers unprecedented power."

Customers using Force CPU-21 and CPU-26 CPUs can employ the CPU-32 with full compatibility yet much higher performance. The new board provides a direct software upgrade path that requires no changes to application programs.

Features Go Beyond Single Board Expectations

The 68030 microprocessor is approximately twice as powerful as its predecessor, the 68020. It imparts the following capabilities to the CPU-32:

- ***Built-in paged memory management unit (PMMU) based on the 68851;***
- ***256 byte data cache in addition to instruction caching for improved data flow;***
- ***Up to a 200% improvement in execution speed of the instruction set;***
- ***Pipelined CPU architecture speeds integrated memory management;***
- ***Burst data and instruction retrieval (16 bytes rather than 4 bytes) boosts throughput.***

The CPU-32's static RAM space will permit 4Mbyte when higher density devices become available. Devices with access times of 100ns allow significant cost savings. For highest performance, SRAMs with 35ns access time are being coupled with the 30MHz versions to provide *constant* zero wait state access. The 20 and 25MHz versions provide constant 1 wait state access.

The CPU-32 also offers two serial ports based on the 68561 multiprotocol communications controller, two parallel interface/timers based on the 68230, two bus interrupter modules (for CPU interrupts) based on the 68153, and a real-time clock.

FGA-001 Gate Array Functions

A proprietary Force design, the FGA-001 is a 132-pin CMOS gate array with 1.5 micron feature size. The array consumes less than 120 milliwatts, features remarkably low gate delays (1.4 ns) and is capable of an internal toggle frequency of 200 Mhz and external toggle frequency of 70 MHZ.

VMEbus interface and control functions include DSACK generation, bus error generation, system reset, bus clock and all on-board control logic.

Operating System Kernel Is Included

The CPU-32 is Force Computer's fourth CPU to include VMEPROM. This real-time kernel enables immediate booting and comprehensive operation the moment the board is installed in a system environment. VMEPROM is fully compatible with and contains a rich subset of capabilities from PDOS, a popular real-time system employed with more than 50 CPU boards from nearly two dozen vendors. The kernel includes a PDOS file manager and BIOS modules. Also included is RAM disk support, a screen editor, disk formatting utilities and a rich user interface providing over 75 commands for debugging, kernel and file manager control.

Force plans to port other operating systems and kernels to the CPU-32. These will be announced as they are made available.

MOTOROLA INC.

Microprocessor Products Group
6501 William Cannon Drive West
Austin, Texas 78735-8598

EDITORIAL CONTACT:
Bob King
512/928-6141

READER CONTACT:
Dale Mitchell
512/928-6705

INQUIRY RESPONSE:
Technical Info Center
P.O. Box 52073
Phoenix, AZ 85072

A NEW 8K x 8 CMOS STATIC RANDOM ACCESS MEMORY, THE MCM6064, IS NOW AVAILABLE FROM MOTOROLA

Austin, Texas, November 11, 1987......The Motorola Memory Products Division is offering three speed versions of the MCM6064, 100ns, 120ns AND 150ns. This device is configured as 8K words of 8 bits each and to accommodate a variety of applications, it is available in both standard and low power versions.

Adding to its popularity, the MCM6064 is pin compalible with the 2764 EPROM family. While operating in a fully static mode, no clocking is required for (E1* and E2), the chip enable pins, to function properly. Because there are both active high and active low enable pins on this memory, the flexibility offered to the designer is a significant consideration.

The low power standby mode is entered automaticaly whenever either of these enables is unasserted. The normal mode of operation is resummed when both enables return to their true asserted state.

While in the low power standby mode, the MCM6064 requires only 100 microamperes maximum and as little as 3 microamperes under typical operating conditions. To support low power and battery backed up applications, the MCM60L64 version is available optionally. It requires only 1 microampere maximum and 0.6 microamperes typically when the standby mode is active.

The CMOS silicon-gate technology allows full operation whenever the external power source is set to +5 volts -/+ 10%. A +2.0 volt to +5.5 volt range will insure data retention. TTL compatibility for all inputs and outputs is guaranteed throughout. Additionally, the outputs can assume a three state condition.

To order either of these memory devices, the 500 piece prise is listed below:

MCM6064P10	$4.03
MCM6064P12	3.70
MCM6064P15	3.36
MCM60L64P10	4.64
MCM60L64P12	4.25
MCM60L64P15	3.86

MOTOROLA ANNOUNCES A NEW M68HC11 DEVICE, THE MC68HC11E9

Austin, Texas, November 13, 1987.....The Motorola Microcontroller Division is pleased to announce the arrival of the MC68HC11E9. This HCMOS single-chip microcontroller will contain a full complement of the existing features of the MC68HC11A8 version. To support more memory intense applications, it comes equipped with 12K Bytes of ROM, 512 Bytes of RAM and 512 Bytes of EEPROM. Depending on the specific need, there are either 3/5 or 4/4 input capture/output compare timer options available. An added security feature is an EEPROM block which protects the EEPROM from accidental erasure. Additionally, the 68HC11E9 features an 8 channel A/D converter and a COP watchdog.

The MC68HC11E9 will be available in a 52-pin PLCC. The guaranteed operating temperature range is from -40 to 85 degrees C.

There is a $5300.00 mask charge for this device. ROM patterns are being accepted in December 1987 with sampling scheduled for 1Q88. A ROMless version, the MC68HCE1, will also be available 1Q88 for prototype purposes. Full scale production of the 'E9 will occur during 3Q88. In 1988, the E1 and E9 1,000 unit pricing is $22.04 and $22.65 respectively. The M68HC11EVM will provide development support ($500.00).

MOTOROLA INC.

Microprocessor Products Group
6501 William Cannon Drive West
Austin, Texas 78735-8598

EDITORIAL CONTACT:
Bob King
512/928-6141

READER CONTACT:
Laura Tolpen
512/440-2035

INQUIRY RESPONSE:
Technical Info Center
P.O. Box 52073
Phoenix, AZ 85072

MOTOROLA ANNOUNCES A NEW M68HC05 MCU CORE MEMBER WITH EEPROM, THE MC68HC05B SERIES

Austin, Texas, November 13, 1987.....Motorola's Microcontroller Division announces the HCMOS MC68HC05B series, a new high-performance M68HC05 Microcomputer (MCU) Family member. This series integrates a variety of versatile functions including EEPROM, an A/D converter, and flexible timers. This MCU series is offered in either a 52-lead PLCC or a 48-pin DIP.

Initial members of the 68HC05B family include the following memory configurations:

MC68HC05B4 - 4K ROM AND 176 BYTES OF RAM
MC68HC05B6 - 6K ROM, 256 BYTES OF EEPROM AND 176 BYTES OF RAM
MC68HC805B6 - 6K EEPROM AND 176 BYTES OF RAM

The series core is that of the well known M68HC05. The instruction set includes powerful 8 x 8 multiply, bit set, bit clear, logical and arithmetic operations. An 8-channel A/D converter is provided using the successive approximation technique with full capacitive charge redistribution method results in inherently monotonic and accurate 8-bit conversions. To save power, the A/D converter amplifier power supply can be switched off under software control.

The MC68HC05B6 contains 6K of ROM, whereas the MC68HC05B4 has 4K bytes of ROM. A 6K EEPROM (MC68HC805B6) emulator part will be available in 2Q88 and is a pin for pin replacement for the ROM versions. The 68HC805B6 is ideal for low quantity trial and code qualification phases. All HC05B devices contain an on-board self test program useful for customer incoming inspections and callable routines. All devices in the series contain 176 bytes of RAM.

A 16-bit free running timer is provided with 2 associated 16-bit compare and two 16-bit capture registers. Individual interrupt vectors are provided for timer capture and compare functions. The timer unit also has a two channel pulse width modulation subsystem.

There are 24 bidirectional I/O ports and 8 input only ports. Memory mapped CPU access of these ports allow individual bit manipulation and branch-on-status features. There are 7 interrupts, 6 hardware, and one software. One of the hardware interrupts is external.

The MC68HC05B6 has 256 bytes of byte erasable EEPROM with an on-board charge pump that generates the high voltage necessary for programming and erasing. Typical endurance is 10,000 cycles.

The Serial Communications Interface (SCI) is a very versatile, fully independent, bi-directional, asynchronous receive/transmit unit providing a simple but powerful standard serial link. Noise checking, overrun, and frame error information is provided to the CPU.

There is a programming divider that allows reduction of the internal operating frequency by a factor of 16, which in turn results in a similar reduction of the power dissipation. Two additional power saving modes are stop and wait. These are software controllable modes. A watchdog feature provides a system reset if the watchdog counter is not refreshed within a preset period of time. This subsystem provides for fail-safe operation.

Motorola provides several development and emulation tools to support this new series: the HDS 300 based development system and a cost effective evaluation module (M68HC05EVM).

In 1989, the 68HC05B6 will be available for $7.50 in 10K quantities and the 68HC05B4 for $3.90 in 10K quantities. The MC68HC805B6 version will be priced at $49.50 and is available in limited quantities starting 2Q88.

EDITORIAL CONTACT:
Angela Hatfield
(602) 952-3613

READER CONTACT:
Technical Operations
(602) 244-7579

INQUIRY RESPONSE:
Technical Information Center
P.O. Box 52073
Phoenix, AZ 85072

MC68030 AUDIO CASSETTE COURSE PREPARES USER FOR DESIGN-INS

Phoenix, Arizona, November 16, 1987 . . . Motorola Semiconductor Products Sector Technical Operations' new MC68030 Audio Cassette Course: An Introduction to the MC68030 32-Bit Microprocessor (MTTA3) prepares the user to design with the newest 32-bit microprocessor in the industry. The self-paced audio cassette course contains three tapes (three and one-half hours long), fully illustrated course notes and related support literature. The course covers the major features of the MC68030 including data cache, burst mode, synchronous bus, and the Internal Memory Management Unit. Upon successful completion, the student will have a working technical knowledge of the MC68030.

The course material is designed in a modular fashion with clearly stated objectives, comprehensive exercises, self-evaluations for each module, and supplemental study references to enable the learner to customize his learning experience. The course is intended to enable a design engineer or programmer already familiar with earlier MC68000 and MC68020 microprocessor family members to successfully use the enhanced MC68030. Motorola's MC68020 (MTTA2) audio cassette course or equivalent experience is a required prerequisite.

The MTTA3 (68030) audio cassette course retails for $125.00. Pricing is in U.S. dollars for U.S. delivery only. Local sales tax should be added to all orders. For Canadian orders or information, call (416) 497-8181. Other locations should contact their nearest Motorola sales office.

The audio cassette course highlights the same technical information as the formal instructor-led course which will be offered in the first half of 1988. To receive a catalog describing all of Motorola Technical Operations' course offerings and to order a copy of the MC68030 audio cassette course, call Motorola Technical Operations at 1-800-521-6274.

Motorola Semiconductor Products Sector Technical Operations offers a wide selection of technical courses ranging from courses for non-technical people to courses for systems integrators, from introductory courses on 8-bit MPU design to advanced courses on 32-bit microprocessors. Motorola Technical Training courses are scheduled in training centers in the United States, Canada, and Europe, and are also offered in South America and the Asia-Pacific Crescent.

ATARI CALL

As most of you know, we are very sensitive to your wishes, as concerns the content of these pages. One of the things that many of you have repeatedly written or called about is coverage for the Atari™ series of 68000 computers.

Actually we haven't been too keen on these systems due to a lack of serious software. They were mainly expensive "game-toy" systems. However, recently we are seeing more and more honest-to-goodness serious software for the Atari machine. That makes a difference. I feel that we are ready to start some serious looking into a section for the Atari computers. Especially since OS-9 is now running on the Atari (review copy on the way for evaluation and report to you). Many of you are doing all kinds of interesting things on these systems. By sharing we all benefit.

This I must stress - *Input from you on the Atari*. As most of you are aware, we are a "contributor supported" magazine. That means that YOU have to do your part. This is the way it has been for over 10 years. We need articles, technical reviews of hardware and software, programming (all languages) and the many other facets of support that we have pursued for these many years. Also I will need several to volunteer to do regular columns on the Atari systems. Without constant input we can't make it fly! So, if you do your part, we certainly will do ours. How about it? Drop me a line or give me a phone call and I will get additional information right back to you. We need your input and support if this is to succeed!

DMW

Classifieds As Submitted - No Guarantees

AT&T 7300 UNIX PC, UNIX V OS, 1MB Memory, 20 MB Hard Disk, 5" Drive, Internal Modem, Mouse. Best Offer Gets It.
S+ System with Cabinet, 20 Meg Hard Disk & 8" Disk Drive with DMAF3 Controller Board. 1-X12 Terminal $4800.
DAISY WHEEL PRINTERS
Qume Sprint 9 - $900
Qume Sprint 5 - $800.
HARD DISK 10 Megabyte Drive - Seagate Model #412 $275.
3 - Dual 8" drive enclosure with power supply. New in box. $125 each.
5 - Siemens 8" Disk Drive, $100 each.
Tano Outpost II, 56K, 2 5" DSDD Drives, FLEX, MUMPS, $495.
TELETYPE Model 43 PRINTER - with serial (RS232) interface and full ASCII keyboard. $250 ready to run.
SWTPC S/09 with Motorola 128K RAM, 1-MPS2, 1-Paralell Port, MP-09CPU Card - $900 complete.
Tom (615) 842-4600 M-F 9AM to 5PM EST

•••

GIMIX SYSTEM: 2 DSDD 8" and 2 SSDD 5" drives. Gimix disk card, 64K RAM print buffer and 2 RS232 ports. Flex OS9 with lots of useful software.
Best offer. 414-442-5143

U.S. Postal Service Statement of Ownership, Management and Circulation (Required by 39 U.S.C. 3685): 1A. Title of Publication: 68 Micro Journal 1B. Publication no: 468510: 2. Date of Filing: 10-01-85: 3. Frequency of Issue: Monthly: 3A. No. of Issues Published Annually: 12. 3B Annual Subscription Price: $24.50 4 and 5: Complete Mailing Address of Known Office, Headquarters or General Business Offices of the Publisher: 5900 Cassandra Smith Rd., Hixson, TN, 37343 6. Full Name and Complete Mailing Address of Publisher: Donald M. Williams Sr., 5900 Cassandra Smith Rd., Hixson, TN 37343. Editor: Donald M. Williams Sr., 5900 Cassandra Smith Rd., Hixson, TN 37343., Managing Editor: Larry E. Williams, 5900 Cassandra Smith Rd., Hixson, TN 37343 7. Owner: Computer Publishing Inc., 5900 Cassandra Smith Rd., Hixson, TN 37343. whose Stockholders are: Donald M. Williams, Sr., Frances J. Williams, Larry E. Williams, Mary E. Robertson, Thomas E. Williams. 8. Known Bondholders, Mortgagees, and other Security Holders Owning or Holding 1 Percent or more of Total Amount of Bonds, Mortgages or other Securities: None. 9. For Completion by Nonprofit Organizations Authorized to Mail at Special Rates: N/A 10. Extent and Nature of Circulation: A. Total No. Copies (Net Press Run): Average No. Copies Each Issue During Preceding 12 Months: 8175., Actual No. Copies of Single Issue Published Nearest to Filing Date: 8500. B. Paid and/or Requested Circulation: 1. Sales Through Dealers And Carriers, Street Vendors And Counter Sales: Average No. Copies Each Issue During Preceding 12 Months: 4999. Actual No. Copies Of Single Issue Published Nearest to Filing Date: 5113. 2. Mail Subscriptions: Average No. Copies Each Issue During Preceding 12 Months 2863. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 2924. C. Total Paid Circulation: Average No. Copies Each Issue During Preceding 12 Months: 7862. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 8037. D. Free Distribution By Mail, Carrier Or Other Means Including Samples, Complimentary, And Other Free Copies: Average No. Copies Each Issue During Preceding 12 Months: 39. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 58. E. Total Distribution (Sum Of C And D): Average No. Copies Each Issue During Preceding 12 Months: 7901, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 8095. F. Copies Not Distributed: 1. Office Use, Left Over, Unaccounted, Spoiled After Printing: Average No. Copies Each Issue During Preceding 12 Months: 259, Actual No. Copies Of Single Issue Published Nearest To Filing Date: 390. 2. Return From News Agents: Average No. Copies Each Issue During Preceding 12 Months: 15. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 15. G. Total (Sum of E, F1 and 2-Should Equal Net Press Run Shown In A: Average No. Copies Each Issue During Preceding 12 Months: 8175. Actual No. Copies Of Single Issue Published Nearest To Filing Date: 8500. 11. I Certify That The Statements Made By Me Above Are Correct And Complete: (Signed): LARRY E. WILLIAMS, Office Manager, 68 Micro Journal, Computer Publishing, Inc. 12. For completion by publishers mailing at the regular rates: Permission requested.

SK★DOS™

The Generic DOS™ for 68000 applications in

- ★ Industrial Control
- ★ Business Use
- ★ Educational Computing
- ★ Scientific Computing
- ★ Number Crunching
- ★ Dedicated Systems
- ★ Turnkey Systems
- ★ Data Collection
- ★ Single board Computers
- ★ Bus oriented Computers
- ★ Graphics Workstations
- ★ One of a kind Systems
- ★ Advanced Hobbyist Use

SK★DOS is a single user disk operating system for computers using Motorola 32 bit CPUs such as the 68008, 68000, 68010, and 68020. It provides the power of a full DOS, yet is simple and easy to use, and will run on systems from 32K to 16 megabytes. Because **SK★DOS** is easily implemented on a new system, we call it "The Generic DOS" which allows programs written for one system to be run on many others.

SK★DOS comes with over 40 commands and system programs, including a 6809 emulator which allows 68K **SK★DOS** to run application programs and languages developed for 6809 SK★DOS and other systems. Assemblers, editors, and higher level language support are available from third party software vendors and through public domain software.

SK★DOS is available for single copy or dealer sales, as well as OEM licensing. Single copies cost $125 (inquire as to available systems). Extremely attractive OEM licensing terms are also available. An optional Configuration Kit contains a detailed Configuration Manual and two disks of source code for system adaptation, including source code for a system monitor/debug ROM and other programs useful for adapting **SK★DOS** to new systems.

SK★DOS

is available from

STAR-K

SOFTWARE SYSTEMS CORPORATION

P.O. BOX 209 · MT. KISCO, NY 10549 · 914/241-0287
TELEX 510501:6774

INDUSTRIAL PASCAL FOR 68000 AND 6809

PCSK is a package that generates code for a 68000 series processor running on a 68000 development system. It includes the compiler, assembler, linker, host debugger, target debugger, and screen editor all integrated together and controlled by a menu driven shell program. Source code is included for the runtime library and many of the utilities. Host operating systems supported are OS-9/68000 (Microware), PDOS (Eyring Research), and VERSAdos (Motorola).

PXK9 is a package that generates code for a 6809 processor running on a 68000 development system. Includes all of the features of the PCSK package above, except for the host debugger. Host operating system is OS-9/68000.

I WANT IT, WHERE DO I GET IT?

For more information on either of these two products please contact Certified Software, South East Media, or one of our European Licensees.

OEM LICENSEES

Gespac sa, 3, chemin des Aulx, CH-1228 Geneva/Plan-les-Ouates, Switz. TEL (022) 713400, TLX 429989

PEP Elektronik Systeme GmbH, Am Klosterwald 4, D-8950 Kaufbeuren, West Germany. TEL (08341) 8974, TLX 541233

Eltec Elektronik GmbH, Galileo-Galilei-Strasse, 6500 Mainz 42, Postfach 65, West Germany. TEL (06131) 50031, TLX 4187273

DISTRIBUTORS

R.C.S. Microsystems Ltd, 141 Uxbridge Road, Hampton Hill, Middlesex, England. TEL 01-9792204, TLX 8951470

Dr. Rudolf Keil GmbH, Porphystrasse 15, D-6905 Schriesheim, West Germany. TEL 062 03/6741, TLX 465025

Elsoft AG, Zelgweg 12, CH-5405 Baden-Daettwil, Switzerland. TEL 056-833377, TLX 828275

Byte Studio Borken, Butenwall 14, D-4280 Borken, West Germany. TEL 02861-2147, TLX 813343

CERTIFIED SOFTWARE CORPORATION

616 CAMINO CABALLO, NIPOMO, CA 93444
TEL: (805) 929-1395 TELEX: 467013
FAX: (805) 929-1395 (MID-8AM)

SOFTWARE FOR 680x AND MSDOS

SUPER SLEUTH DISASSEMBLERS
EACH $99-FLEX $101-OS9 $100-UNIFLEX
OBJECT-ONLY versions: EACH $50-FLEX,OS9,COCO
interactively generate source on disk with labels, include xref, binary editing
specify 6800,1,2,3,5,8,9/6502 version or Z80/8080,5 version
OS9 version also processes FLEX format object file under OS9
COCO DOS available in 6800,1,2,3,5,8,9/6502 version (not Z80/8080,5) only
68010 disassembler $100-FLEX,OS9,UNIFLEX,MSDOS,UNIX,SKDOS

CROSS-ASSEMBLERS WITH MACRO CAPABILITIES
EACH $50-FLEX,OS9,UNIFLEX,MSDOS,UNIX,SKDOS 3/$100 ALL/$200
specify: 180x,6502,6801/11,6804,6805,6809,Z8,Z80,8048,8051,8085,68010,32000
modular cross-assemblers in C, with load/unload utilities
sources for additional $50 each, $100 for 3, $300 for all

DEBUGGING SIMULATORS FOR POPULAR 8-BIT MICROPROCESSORS
EACH $75-FLEX $100-OS9 $80-UNIFLEX
OBJECT-ONLY versions: EACH $50-COCO FLEX,COCO OS9
interactively simulate processors, include disassembly formatting, binary editing
specify for 6800/1, (14)6805, 6502, 6809 OS9, Z80 FLEX

ASSEMBLER CODE TRANSLATORS FOR 6502, 6800/1, 6809

6502 to 6809	$75-FLEX $85-OS9 $80-UNIFLEX
6800/1 to 6809 & 6809 to position ind.	$50-FLEX $75-OS9 $60-UNIFLEX

FULL-SCREEN XBASIC PROGRAMS with cursor control
AVAILABLE FOR FLEX, UNIFLEX, AND MSDOS

DISPLAY GENERATOR/DOCUMENTOR	$50 w/source, $25 without
MAILING LIST SYSTEM	$100 w/source, $50 without
INVENTORY WITH MRP	$100 w/source, $50 without
TABULA RASA SPREADSHEET	$100 w/source, $50 without

DISK AND XBASIC UTILITY PROGRAM LIBRARY
$50-FLEX $30-UNIFLEX/MSDOS
edit disk sectors, sort directory, maintain master catalog, do disk sorts,
resequence some or all of BASIC program, xref BASIC program, etc.
non-FLEX versions include sort and resequencer only

CMODEM TELECOMMUNICATIONS PROGRAM
$100-FLEX,OS9,UNIFLEX,MS-DOS,UNIX,SKDOS
OBJECT-ONLY versions: EACH $50
menu-driven with terminal mode, file transfer, MODEM7, XON-XOFF, etc.
for COCO and non-COCO; drives internal COCO modem port up to 2400 Baud

DISKETTES & SERVICES

5.25" DISKETTES
EACH 10-PACK $7.50-SSSD/SSDD/DSDD
American made, guaranteed 100% quality, with Tyvek jackets, hub rings, and labels

ADDITIONAL SERVICES FOR THE COMPUTING COMMUNITY

CUSTOMIZED PROGRAMMING
we will customize any of the programs described in this advertisement or in our brochure for specialized customer use or to cover new processors; the charge for such customization depends upon the marketability of the modifications.

CONTRACT PROGRAMMING
we will create new programs or modify existing programs on a contract basis, a service we have provided for over twenty years; the computers on which we have performed contract programming include most popular models of mainframes, including IBM, Burroughs, Univac, Honeywell, most popular models of minicomputers, including DEC, IBM, DG, HP, AT&T, and most popular brands of microcomputers, including 6800/1, 6809, Z80, 6502, 680x0, using most appropriate languages and operating systems, on systems ranging in size from large telecommunications to single board controllers; the charge for contract programming is usually by the hour or by the task.

CONSULTING
we offer a wide range of business and technical consulting services, including seminars, advice, training, and design, on any topic related to computers; the charge for consulting is normally based upon time, travel, and expenses.

Computer Systems Consultants, Inc.
1454 Latta Lane, Conyers, GA 30207
Telephone 404-483-4570 or 1717

We take orders at any time, but plan
long discussions after 6, if possible.

Contact us about catalog, dealer, discounts, and services.
Most programs in source: give computer, OS, disk size.
25% off multiple purchases of same program on one order.
VISA and MASTER CARD accepted; US funds only, please.
Add GA sales tax (if in GA) and 5% shipping.

(UNI)FLEX tm Technical Systems Consultants; OS9 Microware;
COCO Tandy; MSDOS Microsoft; SKDOS Star-K Software

K-BASIC™

The Only 6809 BASIC to Binary Compiler for OS-9
FLEX or SK*DOS
Even runs on the 68XXX SK*DOS Systems*

Hundreds Sold at Suggested Retail:

~~$199.00~~

- 6809 - OS-9™ users can now transfer their FLEX™ Extended BASIC (XBASIC) source files to OS-9, compile with the OS-9 version and run them as any other OS-9 binary "CMD" program. Much faster than BASIC programs.

- 6809 - FLEX users can compiler their BASIC source files to a regular FLEX ".CMD" file. Much faster execution.

- 68XXX - SK*DOS™ users running on 68XXX systems (such as the Mustang-08/A) can continue to execute their 6809 FLEX BASIC and compiled programs while getting things ported over to the 68XXX. SK*DOS allows 6809 programs to run in emulation mode. This is the only system we know of that will run both 6809 & 68XXX binary files.

K-BASIC is a true compiler. Compiling BASIC 6809 programs to binary command type programs. The savings in RAM needed and the increased speed of binary execution makes this a must for the serious user. And the price is now RIGHT!

Don't get caught up in the *"Learn a New Language"* syndrome - Write Your Program in BASIC, Debug it in BASIC and Then Compile it to a .CMD Binary File.

For a LIMITED time save over 65%... This sale will not be repeated after it's over! *

SALE SPECIAL:

$69.95

SPECIAL

Thank-You-Sale

Only From:

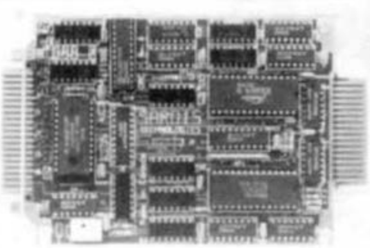
S.E. Media™

CPI

5900 Cassandra Smith Rd.
Hixson, Tn 37343
Telephone 615 842-6809
Telex 510 600-6630

A Division of Computer Publishing Inc.
Over 1,200 Titles - 6800-6809-68000

* K-BASIC will run under 68XXX SK*DOS in emulation mode for the 6809.
Price subject to change without notice.



NEW!

Color Computer "No Halt" Disk Controller

Did you know?

. . . that all the older floppy disk controllers for the CoCo completely tie up (and even halt) the 6809 processor during disk reads and writes? No wonder your keyboard is constantly "losing" characters! Or that your serial port often gives you garbage.

Unleash your CoCo's potential!

Our new Dual Mode Controller (DMC) implements a new "no halt" mode of operation so it can read from or write to disk all by itself. The 6809 is freed to process other tasks and respond to interrupts. This is how OS-9 was meant to run! But the Radio Shack "halt" mode of operation is also retained to maintain full compatibility with existing non-OS9 software.

Other DMC features:

- works with original CoCo, CoCo 2, or CoCo 3 (Multi-Pak required)
- no adjustments — all-digital data separator and write precompensation
- gold plated card-edge connectors for reliability
- ROM socket takes 24 pin or 28 pin chips; dual DOS capability
- Radio Shack DOS 1.1 ROM installed
- 8K bytes cache memory on board (32K b tes optional)
- Includes D.P. Johnson's SDISK package specially modified for the DMC
- fully assembled and tested; 120 day limited warranty

To order: DMC controller with RSDOS 1.1 and SDISK (Level I or II) $149.50 plus $5 S/H ($12 overseas). Terms (prices in $US): check, money order, VISA

(trademarks: OS-9 — Microware and Motorola)

(604) 255-4485 (Pacific Time)
Our ST-2900 SBC is still available,
call or write for catalog/price-list.

SARDIS TECHNOLOGIES
2261 East 11th Ave., Vancouver, B.C., Canada V5N 1Z7

INDUSTRIAL COMPUTER SYSTEMS
68008 - 68020 VME STD SBC
STD BUS I/O OS9

BILL WEST INCORPORATED products and services are intended for industrial users of microcomputers who are looking for a supplier prepared to provide complete engineering and technical support. We configure systems with components and software selected or designed to meet specific customer requirements. We specialize in solving industrial I/O problems. A full range of services is available including system specification, integration, and installation, hardware design and production, and system and application programming.

SYSTEMS

BILL WEST INCORPORATED provides microcomputer systems configured to meet the needs of particular customer applications. These systems use 68000 family processors and the OS9(tm) operating system. We design and configure systems based on VME bus, STD bus, and single board computers, with processors ranging from the 6809 to the 68020. Disk based and ROM based systems are supported. Systems for development and target applications can be supplied.

I/O EXPANSION

The STD bus is an excellent I/O expansion bus. Many off-the-shelf interfaces are available, from simple parallel interfaces and intelligent communication controllers to pneumatic valves and radio receivers on card. BWI interfaces the STD bus to systems via the Motorola I/O channel, the SCSI interface, and ARCNET. The STD bus may be configured as a simple expansion bus, an intelligent subsystem, or a remote I/O system.

MACINTOSH AND ATARI ST EXPANSION

Expansion slots may be added to Macintosh(tm) or Atari ST(tm) systems using the STD bus. Connect an STD expansion bus via the SCSI port, and use any of the wide variety of STD bus interface cards.

BILL WEST INCORPORATED
174 Robert Treat Dr., Milford Connecticut 06460
203 878-9376

DATA-COMP SPECIAL

Heavy Duty Power Supplies

For A limited time our HEAVY DUTY SWITCHING POWER SUPPLY. These are BRAND NEW units. Note that these prices are less than 1/4 the normal price for these high quality units.

Make: Boschert
Size: 10.5 x 5 x 2.5 inches
Including heavy mounting bracket and heatsink

Rating: in 110/220 volts ac (strap change) Out: 130 watts

Output: +5v - 10 amps
+12v - 4.0 amps
+12v - 2.0 amps
-12v - 0.5 amps

Mating Connector: Terminal strip
Load Reaction: Automatic short circuit recovery
SPECIAL: $59.95 each
2 or more $49.95 each
Add: $7.50 each S/H

Make: Boschert
Size: 10.75 x 6.2 x 2.25 inches

Rating: 110/220 ac (strap change) Out: 81 watts

Outputs: +5v - 8.0 amps
+12v - 2.4 amps
+12v - 2.4 amps
+12v - 2.1 amps
-12v - 0.4 amps

Mating Connectors: Molex
Load Reaction: Automatic short circuit recovery
SPECIAL: $49.95 each
2 or more $39.95 each
Add: $7.50 S/H each

5900 Cassandra Smith Rd., Hixson, Tn. 37343 Telephone 615 842-4600 Telex 510 600-6630

THE 6800-6809 BOOKS

..HEAR YE.....HEAR

OS-9™ User Notes

By: Peter Dibble

The publishers of 68' Micro Journal are proud to make available the publication of Peter Dibbles **OS9 USER NOTES**

Information for the BEGINNER to the PRO, Regular or CoCo OS9

Using OS9
HELP, HINTS, PROBLEMS, REVIEWS, SUGGESTIONS, COMPLAINTS, OS9 STANDARDS, Generating a New Bootstrap, Building a new System Disk, OS9 Users Group, etc.

Program interfacing to OS9
DEVICE DESCRIPTORS, DIRECTORIES, "FORKS", PROTECTION, "SUSPEND STATE", "PIPES", "INPUT/OUTPUT SYSTEM", etc.

Programming Languages
Assembly Language Programs and Interfacing; Basic09, C, Pascal, and Cobol reviews, programs, and uses; etc.

Disks Include
No typing all the Source Listings in. Source Code and, where applicable, assembled or compiled Operating Programs. The Source and the Discussions in the Columns can be used "as is", or as a "Starting Point" for developing your OWN more powerful Programs. Programs sometimes use multiple Languages such as a short Assembly Language Routine for reading a Directory, which is then "piped" to a Basic09 Routine for output formatting, etc.

BOOK $9.95

Typeset -- w/ Source Listings
(3-Hole Punched; 8 x 11)
Deluxe Binder - - - - - - - - - - $5.50

All Source Listings on Disk

1-8" SS, SD Disk - - - - $14.95
2-5" SS, DD Disks - - - $24.95

FLEX™ USER NOTES

By: Ronald Anderson

The publishers of 68 MICRO JOURNAL are proud to make available the publication of Ron Anderson's **FLEX USER NOTES**, in book form. This popular monthly column has been a regular feature in 68' MICRO JOURNAL SINCE 1979. It has earned the respect of thousands of 68 MICRO JOURNAL readers over the years. In fact, Ron's column has been described as the 'Bible' for 68XX users, by some of the world's leading microprocessor professionals. The most needed and popular 68XX book available. Over the years Ron's column has been one of the most popular in 68 MICRO JOURNAL. And of course 68 MICRO JOURNAL is the most popular 68XX magazine published.

Listed below are a few of the **TEXT** files included in the book and on diskette.

All TEXT files in the book are on the disks

LOGO C1	File load program to offset memory — ASM PIC
MEMOVE C1	Memory move program — ASM PIC
DUMP C1	Printer dump program — uses LOGO — ASM PIC
SUBTEST C1	Simulation of 6800 code to 6809, show differences — ASM
TERMEM C2	Modem input to disk (or other port input to disk) — ASM
M C2	Output a file to modem (or another port) — ASM
PRINT C3	Parallel (enhanced) printer driver — ASM
MODEM C2	TTL output to CRT and modem (or other port) — ASM
SCIPKG C1	Scientific math routines — PASCAL
U C4	Mini-monitor, disk resident, many useful functions — ASM
PRINT C4	Parallel printer driver, without PFLAG — ASM
SET C5	Set printer modes — ASM
SETBAS1 C5	Set printer modes — A-BASIC

NOTE: .C1, .C2, etc.=Chapter 1, Chapter 2, etc.

**Over 30 TEXT files included is ASM (assembler)-PASCAL-PIC (position independent code) TSC BASIC-C, etc.

Book only: **$7.95** + $2.50 S/H

With disk: 5" **$20.90** + $2.50 S/H

With disk: 8" **$22.90** + $2.50 S/H

Shipping & Handling $3.50 per Book, $2.50 per Disk set
Foreign Orders Add $4.50 Surface Mail
or $7.00 Air Mail
If paying by check - Please allow 4-6 weeks delivery
* All Currency in U.S. Dollars

Continually Updated In 68 Micro Journal Monthly

Computer Publishing Inc.
5900 Cassandra Smith Rd.
Hixson, TN 37343

VISA

(615) 842-4601
Telex 5106006630

*FLEX is a trademark of Technical Systems Consultants
*OS9 is a trademark of Microware and Motorola
*68' Micro Journal is a trademark of Computer Publishing Inc.

!!! Subscribe Now !!!
68 MICRO JOURNAL

OK, PLEASE ENTER MY SUBSCRIPTION

Bill My: Mastercard □ VISA □

Card #____________________ Exp. Date________

For 1 Year ____ 2 Years ____ 3 Years ____

Enclosed: $____________

Name ______________________________

Street ______________________________

City ________________ State ____ Zip ________

Country______________________________

My Computer Is: ________________

Subscription Rates

U.S.A.: 1 Year $24.50, 2 Years $42.50, 3 Years $64.50

*Foreign Surface: Add $12.00 per Year to USA Price.

*Foreign Airmail: Add $48.00 per Year to USA Price.

*Canada & Mexico: Add $9.50 per Year to USA Price.

*U.S. Currency Cash or Check Drawn on a USA Bank !

68 Micro Journal
5900 Cassandra Smith Rd.
POB 849
Hixson, TN 37343

Telephone 615 842-4600
Telex 510 600-6630

Reader Service Disks

Disk- 1	Filesort, Minicat, Minicopy,Minifms, **Lifetime, **Poetry. **Foodlist, **Diet.
Disk- 2	Diskedit w/ inst.& fixes, Prime, *Prnod, **Snoopy, **Football, **Hexpawn,**Lifetime.
Disk- 3	Cbug09, Sec1, Sec2, Find, Table2, Intext, Disk-exp, *Disksave.
Disk- 4	Mailing Program, *Finddat, *Change, *Testdisk.
Disk- 5	*DISKFIX 1, *DISKFIX 2, **LETTER, **LOVESIGN, **BLACKJAK, **BOWLING.
Disk- 6	**Purchase Order, Index (Disk file indx).
Disk- 7	Linking Loader, Rload, Harkness.
Disk- 8	Crtest, Lanpher (May 82).
Disk- 9	Datecopy, Diskfix9 (Aug 82).
Disk-10	Home Accounting (July 82).
Disk-11	Dissembler (June 84).
Disk-12	Modem68 (May 84).
Disk-13	*Initmf68, Testmf68,*Cleanup,*Dskalign, Help,Date.Txt.
Disk-14	*Init, *Test, *Terminal, *Find, *Diskedit, Init.Lib.
Disk-15	Modem9 + Updates (Dec. 84 Gilchrist) to Modem9 (April 84 Commo).
Disk-16	Copy.Txt, Copy.Doc, Cat.Txt, Cat.Doc.
Disk-17	Match Utility, RATBAS, A Basic Preprocessor.
Disk-18	Parse.Mod, Size.Cmd (Sept. 85 Armstrong), CMDCODE, CMD.Txt (Sept. 85 Spray).
Disk-19	Clock, Date, Copy, Cat, PDEL.Asm & Doc., Errors.Sys, Do, Log.Asm & Doc.
Disk-20	UNIX Like Tools (July & Sept. 85 Taylor & Gilchr st). Dragon.C, Grep.C, LS.C, FDUMP.C.
Disk-21	Utilities & Games - Date, Life, Madness, Touch, Goblin, Starshot, & 15 more.
Disk-22	Read CPM & Non-FLEX Disks. Fraser May 1984.
Disk-23	ISAM, Indexed Sequential file Accessing Methods, Condon Nov. 1985. Extensible Table Driven, Language Recognition Utility, Anderson March1986.
Disk-24	68' Micro Journal Index of Articles & Bit Bucket Items from 1979 - 1985, John Current.
Disk-25	KERMIT for FLEX derived from the UNIX ver. Burg Feb. 1986. (2)-5" Disks or (1)-8" Disk.
Disk-26	Compacta UniBoard review, code & diagram, Burlison March '86.
Disk-27	ROTABIT.TXT, SUMSTEST.TXT, CONDATA.TXT, BADMEN.TXT.
Disk-28	CT-82 Emulator, bit mapped.
Disk-29	**Star Trek
Disk-30	Simple Winchester, Dec.'86 Green.
Disk-31	*** Read/Write MS/PC-DOS (SK*DOS)
Disk-32	Heir-UNIX Type upgrade - 68MJ 2/87
Disk-33	Build the GT-4 Terminal - 68MJ 11/87 Condon.

NOTE:

This is a reader service ONLY! No Warranty is offered or implied, they are as received by 68' Micro Journal, and are for reader convenience ONLY (some MAY include fixes or patches). Also 6800 and 6809 programs are mixed, as each is fairly simple (mostly) to convert to the other. Software is available to cross-assemble all.

* Denotes 6800 - ** Denotes BASIC
*** Denotes 68000 - 6809 no indicator.

8" disk $19.50
5" disk $16.95

Shipping & Handling -U.S.A. Add: - $3.50
Overseas add: $4.50 Surface - $7.00 Airmail

68 MICRO JOURNAL

5900 Cassandra Smith Rd.
Hixson, TN 37343
(615) 842-4600 - Telex 510 600-6630

6809/68008 SINGLE BOARD COMPUTERS

The Peripheral Technology Family of Single Board Computers is a Low-Cost Group Which Ranges From an Entry Level 8-Bit Version to a Powerful 68008-Based Board. A Product is Available to Fit Almost Every User's Requirements.

PT69-5

- 6809 Processor/2MHZ Clock
- 4 RS-232 Serial Ports
- 2 8-Bit Parallel Ports
- 4K-16K EPROM/60K Ram
- Parallel Printer Interface
- DS/DD Controller for 35-80
- Track Drives Ranging From SS/SD-DS/DD
- Winchester Interface Port

Price: $315.00

PT69-3

- 6809 1 MHZ Processor
- 2 RS-232 Serial Ports
- 2 8-Bit Parallel Ports
- 4K EPROM/59K User Ram
- DS/DD Controller for 35-80 Track Drives Ranging From SS/SD-DS/DD

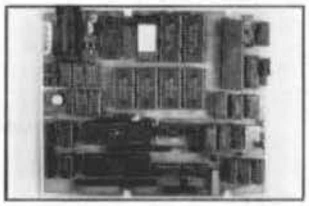
Price: $249.95

OS9 L1 For
PT69 BOARDS: $200.00
SK*DOS: $ 49.95

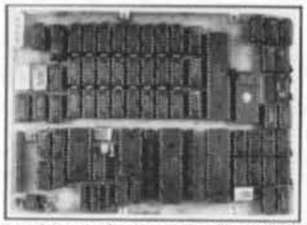
PT68K-1A

- MC68008 12.5 MHZ Processor
- 768K RAM/64K EPROM
- 4-RS-232 Serial Ports
- Winchester Interface Port
- Floppy Disk Controller for 4 5+" Drives
- 2 8-Bit Parallel Ports

BOARD: $499.00
with Professional OS9: $895.00
with SK*DOS: $595.00



PERIPHERAL TECHNOLOGY
1480 Terrell Mill Road, Suite 870
Marietta, Georgia 30067
(404) 984-0742 Telex # 880584
VISA/MASTERCARD/CHECK/C.O.D.

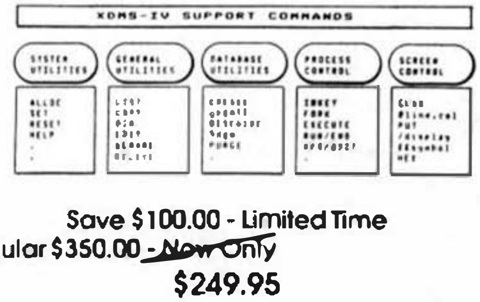


™OS9 Is A Trademark Of Microware and Motorola.

Send For Catalogue For Complete Information On All Products.

XDMS-IV

Data Management System



Save $100.00 - Limited Time
Regular $350.00 - ~~Now~~ Only
$249.95

Technical telephone assistance: Tel 914-941-3552 (Evenings)
FLEX™ Tecnical Systems Consultants, SK•DOS™ STAR-KITS Corp.

FOR 6809 FLEX-SK-DOS(5/8")

Up to 32 groups/fields per record! Up to 12 character filed name! Up to 1024 byte records! User defined screen and print control! Process files! Form files! Conditional execution! Process chaining! Upward/Downward file linking! File joining! Random file virtual paging! Built in utilities! Built in text line editor! Fully session oriented! Enhanced forms! Boldface, Double width, Italics and Underline supported! Written in compact structured assembler! Integrated for FAST execution!

XDMS-IV Data Management System

XDMS-IV is a brand new approach to data management. It not only permits users to describe, enter and retrieve data, but also to process entire files producing customized reports, screen displays and file output. Processing can consist of any of a set of standard high level functions including record and field selection, sorting and aggregation, lookups in other files, special processing of record subsets, custom report formatting, totaling and subtotaling, and presentation of up to three related files as a "database" on user defined output reports.

POWERFUL COMMANDS!

XDMS-IV combines the functionality of many popular DBMS software systems with a new easy to use command set into a single integrated package. We've included many new features and commands including a set of general file utilities. The processing commands are Input-Process-Output (IPO) oriented which allows almost instant implementation of a process design.

SESSION ORIENTED!

XDMS-IV is session oriented. Enter "XDMS" and you are in instant command of all the features. No more waiting for a command to load in from disk! Many commands are immediate, such as CREATE (file definition), UPDATE (file editor), PURGE and DELETE (utilities). Others are process commands which are used to create a user process which is executed with a RUN command. Either may be entered into a "process" file which is executed by an EXECUTE statement. Processes may execute other processes, or themselves, either conditionally or unconditionally. Menus and screen prompts are easily coded, and entire user applications can be run without ever leaving XDMS-IV!

IT'S EASY TO USE!

XDMS-IV keeps data management simple! Rather than design a complex DBMS which hides the true nature of the data, we kept XDMS-IV file oriented. The user view of data relationships is presented in reports and screen output, while the actual data resides in easy to maintain files. This aspect permits customized presentation and reports without complex redefinition of the database files and structure. XDMS-IV may be used for a wide range of applications from simple record management systems (addresses, inventory ...) to integrated database systems (order entry, accounting...). The possibilities are unlimited...

Visa & Master Card Excepted

Telephone: 615-842-4601 or Telex: 510 600-6630
Or Write: S.E. Media, 5900 Cassandra Smith Rd.,
Hixson, Tenn. 37343

THE GMX 020BUG DEBUGGER/DIAGNOSTIC PACKAGE

This extensive firmware package provides a broad range of program development tools and a complete suite of diagnostic programs for exercising GMX Micro-20 hardware.

The debugger includes commands for displaying and modifying registers and memory. If the optional 68881 Floating-Point Coprocessor is installed, its registers are also accessible. Memory can be displayed in hexadecimal and ASCII format, as floating-point values (single, double, extended or packed format), or as disassembled instructions (including FPC instructions). Memory modify can be done with hexadecimal values, with ASCII strings, with floating-point values, or with a one-line assembler which supports the full 68020 instruction set (although not the FPC instructions). Block move, fill, and search are also available.

Several different modes for tracing or executing user programs are provided, along with a powerful breakpoint facility. Programs and data may be downloaded from a host system or uploaded back to the host, and the GMX Micro-20 console may be used as a host system terminal. A serial printer may be hooked up, and used to make hardcopy listings of debugger sessions as desired.

The diagnostic firmware includes 90 test commands and 16 utilities. Complete test suites are provided for each functional block of the GMX Micro-20's hardware, including, for example, 9 different tests for memory, 9 tests for serial I/O ports, 2 tests for the 68881 FPC, and 9 tests for the optional memory management unit. For the peripheral control interfaces (floppy disk, SASI/SCSI hard disk or tape), test commands support a broad range of peripheral operations (read, write, format, etc.) so that the user may test both the interface and an attached device. Tests are provided for add-on I/O boards, including the ARCnet interface, I/O Channel interface, and parallel and serial expansion boards.

The utility commands allow the user to execute groups of test commands conveniently, repeat commands or command groups, enable or disable detailed fault reporting, count detected errors, or execute all the non-peripheral tests as a group. A switch option allows this last function to be invoked automatically at power-on or reset. Other utilities allow the user to check the state of the various jumpers and switches on the GMX Micro-20 directly.

In addition to the Diagnostic command package, 020Bug contains a confidence test which is always run after power-on or reset. This test does a quick checkout of the processor and the basic system elements that are needed for 020Bug operation. If any defect is found, an error code is signalled by on-and-off blinks of an LED.

DEBUGGING COMMANDS

MD — Memory display
MM — Memory modify
MS — Memory set
BF — Block fill
BM — Block move
BS — Block search
RD — Register display
RM — Register modify

OF — Offset registers
BR — Breakpoint set
NOBR — Breakpoint delete
G — Go to target code
GD — Go, delete breakpoints
GN — Go, stop after 1 instruction
GT — Go, set temp breakpoint
T — Trace
TC — Trace on change of flow
TT — Trace to temp breakpoint
LO — Download
DU — Upload
VE — Verify download
TM — Terminal mode
PA — Printer attach
NOPA — Remove printer
PF — Port format
TD — Time display
TS — Time set
SD — Switch directory
RS — Restart system
OS — Boot operating system

UTILITY COMMANDS

NV — Non-verbose mode
SE — Stop on error mode
LE — Loop on error mode
LC — Loop continual mode
ST — Selftest mode
STL — Selftest with LED mode

DE — Display errors
ZE — Zero errors
DP — Display pass count
ZP — Zero pass count
RL — Read loop
WL — Write loop
DJ — Display baud rate jumper settings
DS — Display switch
MJ — Display MMU board jumper settings
SX — Scan I/O expansion space

TEST COMMANDS

AN — ARCnet interface tests
A — Wakeup test
B — DIP Switch test
C — Interrupts test
D — Buffer test

CA20 - On chip cache tests
A — Basic caching
B — Unlike function codes
C — Disable
D — Clear

FD — Floppy disk tests
A — Set parameters
B — Drive select toggle
C — Side select toggle
D — Restore
E — Seek
F — Format track
G — Read
H — Write
I — Copy buffer
J — Compare buffer
K — Fill buffer

IC — I/O Channel tests
A — Print test pattern
B — Bit rotate

MH — Miscellaneous hardware tests
A — 68881 FPC instructions
B — 68881 FPC control functions
C — Tick generator
D — Interrupt sources

MT — Memory tests
A — Set function code
B — Set start address
C — Set end address
D — Random instruction test
E — March address test
F — Walk-a-bit test
G — Refresh test
H — Random byte test
I — Program test
J — TAS test
K — Test 0000-1FFF
L — Partial longword writes test

MU — Memory Management tests
A — Map RAM data test
B — Map RAM address test
C — Map RAM partial write test
D — Map RAM random data test
E — Accessed bit reset test
F — Address mapping test
G — Accessed/Dirty bits test
H — Valid/Write Enable test
I — Task size test

PP — Parallel port tests
A — Print test pattern
B — Continual test bit pattern
C — Test bit pattern for 10 sec

PX — Parallel I/O expansion board tests
A — Data, handshake, and IRQ test
B — P4 connector test
C — Data and hand shake toggle

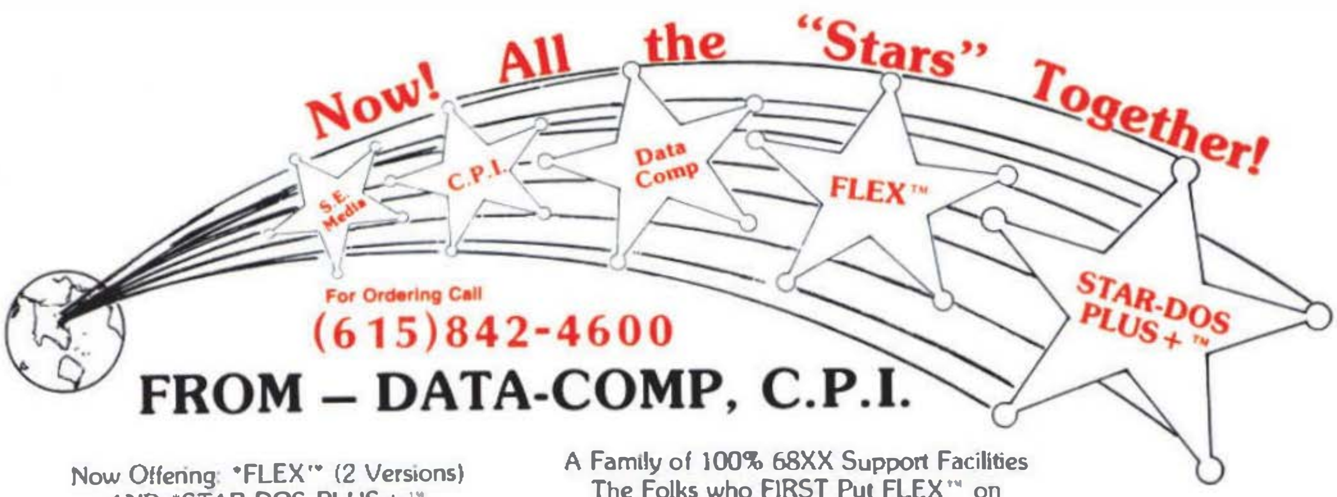
SA — SASI/SCSI port with SASI device
A — Select drive
B — Scan data lines
C — Restore
D — Seek
E — Read
F — Write
G — Compare buffers
H — Fill write buffer
I — Test interrupt
J — Park head
K — Format

SC — SASI/SCSI port with SCSI device
A — Select drive
B — Scan data lines
C — Restore
D — Seek
E — Read
F — Write
G — Compare buffers
H — Fill write buffer
I — Test interrupt
J — Stop drive
K — Format

SI — Serial I/O tests
A — Select DUARTs
B — Internal loopback
C — External loopback
D — Baud rates
E — Parity modes
F — Character lengths
G — Handshake lines
I — BREAK detect
J — Interrupt output
K — Continual handshake toggle

TA — Tape drive tests
A — Rewind
B — Read
C — Write
E — Compare buffers
F — Fill write buffer
G — Erase

GMX™ 1337 W. 37th Place, Chicago, IL 60609
(312) 927-5510 — TWX 910-221-4055 — FAX (312) 927-7352



For Ordering Call
(615)842-4600

FROM – DATA-COMP, C.P.I.

Now Offering: *FLEX™ (2 Versions)
AND *STAR-DOS PLUS+™

A Family of 100% 68XX Support Facilities
The Folks who FIRST Put FLEX™ on
The CoCo

FLEX-CoCo Sr.
with TSC Editor
TSC Assembler
Complete with Manuals
Reg. $250.00 **Only $79.95**

STAR-DOS PLUS+
- Functions Same as FLEX
- Reads - writes FLEX Disks $34.95
- Run FLEX Programs
- Just type: Run "STAR-DOS"
- Over 300 utilities & programs to choose from.

FLEX-CoCo Jr.
without TSC
Editor & Assembler
$49.95

PLUS

ALL VERSIONS OF FLEX & STAR-DOS• INCLUDE

TSC Editor
Reg $50.00
NOW $35.00

+ Read-Write-Dir RS Disk
+ Run RS Basic from Both
+ More Free Utilities
+ External Terminal Program
+ Test Disk Program
+ Disk Examine & Repair Program
+ Memory Examine Program
+ Many Many More!!!

TSC Assembler
Reg $50.00
NOW $35.00

CoCo Disk Drive Sygstems

2 THINLINE DOUBLE SIDED DOUBLE DENSITY DISK DRIVES SYSTEM WITN POWEB SUPPLY,CABINET,DISK DBIVE CABLE,J&M NEW DISK CONTROLLER JPD-CP WITH J-DOS,RS-DOS OPERATING SYSTEMS.	$469.95

* Specify What CONTROLLER You Went J&M, or RADIO SHACK

THINLINE DOUBLE SIDED DOUBLE DENSITY 40 TRACKS	$129.95

Verbatim Diskettes

Single Sided Double Density	$ 24.00
Double Sided Double Density	$ 24.00

Controllers

J&M JPD-CP WITH J-DOS	$139.95
WITH J-DDS, RS-DOS	$159.95
RADIO SHACK 1.1	$134.95
RADIO SHACK Disk CONTROLLER 1.1	$134.95

Disk Drive Cables

Cable for One Drive	$ 19.95
Cable for Two Drives	$ 24.95

MISC

64K UPGRADE FOR C,D,E,F, AND COCO II	$ 29.95
RADIO SHACK BASIC 1.2	$ 24.95
RADIO SHACK DISK BASIC 1.1	$ 24.95
DISK DRIVE CABINET FOR A SINGLE DRIVE	$ 49.95
DISK DRIVE CABINET FOR TWO THINLINE DRIVES	$ 69.95

PRINTERS

EPSON LX-80	$289.95
EPSON MX-70	$125.95
EPSON MX-100	$495.95

ACCESSORIES FOR EPSON

8148 2K SERIAL BOARD	$ 89.95
8149 32K EXPAND TO 128K	$169.95
EPSON MX-RX-80 RIBBONS	$ 7.95
EPSON LX-80 RIBBONS	$ 5.95
TRACTOR UNITS FOR LX-80	$ 39.95

CABLES & OTHER INTERFACES
CALL FOR PRICING

DATA-COMP
5900 Cassandra Smith Rd.
Hixson, TN 37343

SHIPPING
USA ADD 2%
FOREIGN ADD 5%
MIN. $2.50

☎ (615)842-4600
For Ordering
Telex 5106006630

An Ace of a System in Spades! The New

MUSTANG-08/A™

Now with 4 serial ports standard & speed increase to 12 Mhz CPU + on board battery backup and includes the PROFESSIONAL OS-9 package - including the $500.00 OS-9 C compiler! This offer won't last forever!

NOT 128K, *NOT* 512K

FULL 768K No Wait RAM

The **MUSTANG-08™** system took every hand from all other 68008 sysems we tested, running OS-9 68K!

The **MUSTANG-08** includes OS9-68K™ and/or Peter Stark's SK*DOS™. SK*DOS is a single user, single tasking system that takes up where *FLEX™ left off. SK*DOS is actually a 68XXX FLEX type system *(Not a TSC product.)*

The OS-9 68K system is a full blown multi-user, multi-tasking 68XXX system. All the popular 68000 OS-9 software runs. It is a speed whiz on disk I/O. Fact is: the MUSTANG-08 is faster on disk access than some other 68XXX systems are on memory cache access. Now, that is fast! And that is just a small part of the story! See benchmarks!

SALE!

System includes OS-9 68K or SK*DOS - Your Choice

Specifications:

CPU	MC68008	12 Mhz
RAM	768K	256K Chips
	No Wait States	
PORTS	4 - RS232	MC68681 DUART
	2 - 8 bit Parallel	MC6821 PIA
CLOCK	MK48T02	Real Time Clock Bat. B/U
EPROM	16K, 32K or 64K	Selectable
FLOPPY	WD1772	5 1/4 Drives
HARD DISK	Interface Port	WD1002 Board

Now more serial ports - faster CPU Battery B/U - and $850.00 OS-9 Professional with C compiler included!

***$400.00** See Mustang-02 Ad - page 5 for trade-in details

Mustang Hi-Speed Systems
Only from Data-Comp Div.
68008-68020

MUSTANG-08

LOOK

	Seconds	32 bit Integer	Register Long
Other 68008 8 Mhz OS-9 68K	...	18.0	...9.0
MUSTANG-08 10 Mhz OS-9 68K	....	9.8	...6.3

C Benchmark Loop

```
Main()
    {
        /* int I; */
        register long I;
        for (I=0; I < 999999; ++I);
    }
```

Now even faster! with 12 Mhz CPU

C Compile times: OS-9 68K Hard Disk		
MUSTANG-08	8 Mhz CPU	0 min - 32 sec
Other popular 68008 system		1 min - 05 sec
MUSTANG-020		0 min - 21 sec

♠ 25 Megabyte Hard Disk System

$2,398.90

Complete with PROFESSIONAL OS-9 includes the $500.00 C compiler, PC style cabinet, heavy duty power supply, 5" DDDS 80 track floppy, 25 MegByte Hard Disk - Ready to Run

Unlike other 68008 systems there are several significant differences. The MUSTANG-08 is a full 12 Megahertz system. The RAM uses NO wait states, this means full bore MUSTANG type performance.

Also, allowing for addressable ROM/PROM the RAM is the maximum allowed for a 68008. The 68008 can only address a total of 1 Megabytes of RAM. The design allows all the RAM space (for all practical purposes) to be utilized. What is not available to the user is required and reserved for the system.

A RAM disk of 480K can be easily configured, leaving 288K free for program/system RAM space. The RAM DISK can be configured to any size your application requires (system must have 128K in addition to its other requirements). Leaving the remainder of the original 768K for program use. Sufficient source included (drivers, etc.)

FLEX is a trademark of TSC

MUSTANG-08 is a trademark of CPI

Data-Comp Division

A Decade of Quality Service"
Systems World-Wide

Computer Publishing, Inc. 5900 Cassandra Smith Road
Telephone 615 842-4601 - Telex 510 600-6630 Hixson, Tn 37343

* Those with SWTPC hi-density FLEX 5" - Call for special info.